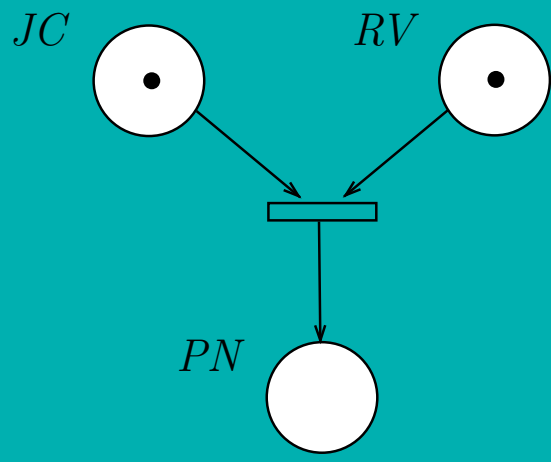
Petri nets

Janette Cardoso, Robert Valette

PETRI NETS



JANETTE CARDOSO
ROBERT VALETTE

Janette Cardoso

Robert Valette

Petri Nets

First revised and augmented edition, translated by
J. Cardoso and R. Valette
from
Redes de Petri
J. Cardoso and R. Valette
Editora da UFSC, ISBN 85-328-0095-5.

For Malik, Thomas and Aïda
Janette

For Marly, Fabien and Aline
Robert

Contents

List of Figures

List of Examples

Foreword

Since their introduction nearly 60 years ago, Petri nets have proven to be an invaluable tool in the realm of computer science, system engineering, and various other disciplines. Their versatility and unambiguous semantics have led to widespread application across diverse domains, including the modelling and optimisation of manufacturing systems, the formal verification of network protocols, the modelling of business process management or biological systems, ... The enduring relevance and adaptability of Petri nets underscore their significance in both academic research and practical applications.

While there is a large literature dedicated to the theory of Petri nets, and their application in the study of discrete event systems, there are relatively fewer introductory resources tailored to engineering students. This book aims to fill this void by providing an accessible introduction, designed for those beginning their journey in this field, or interested in finding out more about this subject of limitless exploration.

Although Petri nets represent the authors' primary research domain, this book is specifically designed as a teaching resource. The design and the content selection may be better understood by looking at the long and rich history of this book. Indeed, the current text is to a large extent an improved and extended version of a course that has been taught in several engineering schools and universities in Toulouse for the last 30 years. The original Portuguese version of this book, published in 1997, was primarily based on handouts written by Robert Valette for his lectures at INSA and ENSEEIHT during the 1990s. Widely used in graduate courses in Brazil, the first edition quickly sold out. In response to high demand, the authors made a free PDF version available online, which was subsequently redistributed by numerous websites. Since then, parts of this material have been used in graduate and undergraduate courses at Toulouse Capitole University and at ISAE-SUPAERO, in the form of lecture slides, available in both French and English.

Through clear and concise presentations, this book introduces the core concepts underlying Petri nets and their use in modeling and formal verification. It also prepares readers for more advanced topics and practical applications, reflecting the evolution and maturation of Petri nets as a fundamental concept in modeling and analyzing concurrent systems.

This new English edition provides additional examples and is updated to reflect years of teaching experience. In the following chapters, you will explore the basic concepts and formalisms of Petri nets, delve into advanced topics such as reachability analysis and performance evaluation, and examine real-world case studies that demonstrate the practical application of Petri nets. The book also includes numerous examples, exercises, and illustrations to enhance your learning experience and deepen your understanding of the material.

Compared to previous editions, several new sections have been added, covering topics such as:

- How to start designing a Petri net;
- How to simulate a Petri net and study its behavior;
- Discussion about validation and verification techniques using Petri nets;
- Methods for modeling large systems using a modular, compositional approach.

In addition, three new appendices, exploring more specialized subjects, have been added:

- Appendix C presents a comprehensive application example that illustrates the concepts introduced in Part 1, including modeling from scratch, formal verification, and modular modeling.
- Appendix D provides detailed guidance on performing synchronous and asynchronous composition of Petri net models.
- Appendix E introduces TINA, a freely available toolbox that includes a Petri net editor and several analysis tools.

I hope that this book will inspire you to appreciate the elegance and power of Petri nets and motivate you to apply them in your own projects and research.

Happy learning!

Silvano dal Zilio

Toulouse
Septembre 2024

Preface to the Portuguese Edition

History

The Petri net is a graphical and mathematical tool that is well-suited to a wide range of applications where notions of events and simultaneous evolutions are important.

This theory is quite young, originating from the thesis titled Communication with Automata, defended by Carl Adam Petri in 1962 at the University of Darmstadt, Germany. Carl Adam Petri, born in 1926 in Leipzig, is a professor at the University of Bonn. Anatol W. Holt was captivated by this work, and under his influence, a group of researchers at the Massachusetts Institute of Technology-MIT, in the United States, laid the foundations between 1968 and 1976 of what became known as Petri nets. Among these pioneers were F. Commoner and M. Hack.

Applications of Petri nets include performance evaluation, formal analysis, and verification in discrete systems, communication protocols, manufacturing workshop control, real-time and/or distributed software design, information systems (such as organizational management), transportation systems, logistics, database management, human-machine interfaces, and multimedia.

Regarding the control of automated manufacturing systems, the application of Petri nets initially took place in France, in a slightly modified form known as the Grafcet standard for programming industrial programmable logic controllers (PLCs). This standard was first proposed by a commission of the Afcet (Association française des sciences et technologie de l'information et des systèmes) in 1977 and became an industrial standard (C03.190-UTE) in France in 1980, later becoming a European standard through the central office of the IEC under reference IEC 848.

The complexity of discrete event systems, particularly in the case of automated manufacturing systems, leads to a hierarchical decomposition with multiple control levels. Typically, five levels are used: planning, scheduling, global coordination, subsystem coordination, and direct control (PLCs directly connected to sensors and actuators). Considering the use of Petri nets at the coordination level, the most active countries are Germany (PSI Society in Berlin) and Japan (Hitachi Society). In France, IXI Society (Toulouse) is a notable example.

Advantages

The advantages of using Petri nets can be summarized as follows:

- It is possible to describe a partial order between various events, which allows for flexibility;

- Both states and events are explicitly represented;
- A single family of tools is used throughout the specification, modeling, analysis, performance evaluation, and implementation processes;
- A single family of tools is used across the different levels of the hierarchical control structure, facilitating the integration of these levels;
- A precise and formal description of synchronizations becomes possible, which is essential for achieving the necessary operational safety.

Book Organization

This book corresponds to the lecture notes used, since 1991, in the course Real-Time Systems I for the Graduate Program in Electrical Engineering at the Federal University of Santa Catarina.

The book is organized into two parts. The first part covers the basic model, also known as the ordinary Petri net. Chapter 1 provides an introduction, beginning with the definition of discrete event systems, informally situating the model, its importance, and areas of application.

In Chapter 2, the basic aspects of the Petri net model are presented: definitions and properties. How to perform analysis is described in Chapter 3. It is helpful to complement the study of this chapter with the *ARP* software for Petri net analysis, developed at LCMI (Laboratory of Control and Microinformatics) of the Department of Electrical Engineering at UFSC (already distributed to some universities, including in France, Belgium, and England).

The second part of this book deals with the interaction between the Petri net and data, time, and the external environment. Chapter 4 introduces interpreted Petri nets, which allow for the representation of interaction with the external environment. Chapter 5 presents high-level nets, which allow for individual token representation, enabling the token to carry information, thereby increasing the abstraction level of the model. Different ways to incorporate time are addressed in Chapter 6.

Chapter 7 discusses the implementation methods for systems represented by Petri nets.

Chapter 8 addresses more recent applications of the model. It covers the joint use of Petri nets with other theories, such as fuzzy logic and linear logic. The former allows for modeling imprecise markings, representing real situations where information about the system is incomplete. The latter allows for expressing the notion of resources, which classical logic cannot handle. This chapter also discusses the use of Petri nets in hybrid systems, such as batch systems, which contain both continuous parts and parts driven by discrete events.

Throughout the book, applications of Petri nets and their extensions in modeling and implementation will be presented, with a focus on the field of manufacturing automation.

This book can also be used for a one-semester undergraduate course on Petri nets, covering Chapters 1 to 4 and part of Chapters 6 and 7.

Acknowledgments

The authors thank the students and colleagues whose questions, doubts, and suggestions contributed to this work, and Brigitte Pradin-Chézalviel, who participated in the development of the section "Petri Nets as Semantics for Linear Logic".

They also thank UFSC, CAPES, CNPq, and LAAS for making this book possible, and the support from the team at the Laboratory of Control and Microinformatics (LCMI/EEL/UFSC), who helped bring it to fruition.

Special thanks go to Eduardo Souza (esms), who, during the final version, provided invaluable assistance with IT emergencies with professionalism and kindness, to colleagues Carlos Alberto Maziero and Guilherme Bittencourt for their help with LaTeX issues, to Hugo Leonardo Gosmann for assisting with the editing of part of the final version, and to Maria Joana Zucco, Eda Brustolin, and Ana Lúcia do Amaral for their support and speed in the review process.

J. C.
R. V.

Florianópolis, SC
May 1997

Part I

Basic Model

Chapter 1

Vocabulary and concepts

The Petri net model was proposed by Carl Petri to model communication between automata PETRI (1966).

To guide the reader through this area of activity, the characterization of such systems and the basic concepts used in their modeling are presented first. The finite state machine, which is widely used to represent such systems, is discussed, along with its limitations.

Before detailing the formal definition of the Petri net in Chapter 2, this chapter presents an informal introduction. The aim is to familiarize the reader with the concepts, allowing to model simple systems easily. Through this introduction, the power of this modeling tool will be highlighted. Even without detailing the analysis techniques presented in Chapter 3, it will be observed how, modeling by means of Petri nets, aids in detecting specification errors.

Before presenting Petri nets, let us present the different types of existing systems and in particular, the discrete event systems (DES).

1.1 Systems and Modeling

Scientists and engineers are concerned with the development of techniques for designing, controlling, and performance evaluation of systems. In order to do this it is necessary to have a model of this system to describe and understand it. We are interested in dynamic systems, whose behaviour depends on time.

1.1.1 Types of variables

A mathematical model of a system is characterized by:

- the nature of the independent variable representing the time,
- the nature of its state variables.

We generally classify variables into two categories:

- *Continuous variables* are variables that take their values on the domain of reals $\mathcal{R}$.
- *Discrete variables* are variables that take their values on a countable set as the set of natural numbers $\mathcal{N}$, or on sets whose number of elements is finite.

From this classification, dynamic systems can be put in various categories. Let us quickly review them.

1.1.2 Types of systems

Continuous systems: A continuous system includes only continuous variables. Furthermore, time is also a continuous variable. When time is represented by a continuous variable, we say that it is a *dense time*. This represents a large domain of systems modeled by algebraic-differential equations. An example of a continuous system with a single state variable is shown in FIG. 1.1.

Sampled systems: These are systems where time is represented discretely while the state variables remain continuous. The time represented as an infinite series of instants identified by natural numbers
$\theta_0, \theta_1, \ldots, \theta_{n-1}, \theta_n, \theta_{n+1}, \ldots$
It is generally assumed that these instants are equally distributed. The value of a state variable at a given time ($x(\theta)$, for example) is a real number. An example of a sampled system with a single state variable is shown in FIG. 1.2, with the dotted line representing an underlying continuous model.

This modeling technique can be used in two cases:

- The system being modeled (the underlying continuous model) actually includes a sampler that the mathematical model faithfully represents.
- The system being modeled does not include a sampler and is a continuous system, but a coarser representation is chosen by only considering the values of the variables at predetermined times to avoid a precise representation by a complex algebraic-differential system.

The second case corresponds to a *qualitative* vision achieved by discretizing time.

Discrete systems: These systems have a discrete representation of the state associated with continuous (dense) time. This is suitable when the state is naturally discrete. For example, a lamp can be on or off, or a piece of equipment can be functional or faulty. It can also be used for a qualitative view of a system. For example, in an elevator, we may not detail the acceleration and deceleration phases, so the state of the cabin is coded by a finite set of values: { *"stopped at the ground floor"*, *"moving between the ground floor and the first floor"*, *"stopped at the first floor"*, *"moving between the first and second floor"*, ..., *"stopped at the last floor"*}. An example of a discrete system with a single state variable is represented in FIG. 1.3.

With a *qualitative* vision, the main difference compared to the previous case is that the state variables are discretized, but time remains dense. We are interested in the position of the elevator at all the instant of dense time, even if the position is approximate (between two floors). In sampled systems, we are interested in the exact position of the elevator at every θ_i second.

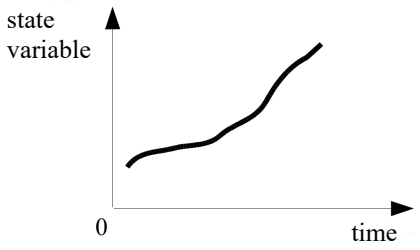


Figure 1.1: Continuous system.

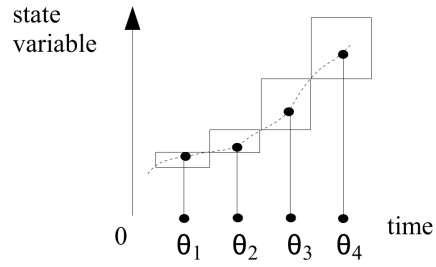


Figure 1.2: Sampled system

Discrete event systems: These systems are represented solely by a series of discrete events. Time is coded by this sequence of events and is thus discretized. The events are discrete because they correspond to state changes between discrete states of the system. Time and states are both discretized. Unlike sampled systems, the events are not equally distributed over time. They correspond to transitions from one qualitative state to another. The time (viewed continuously) between two successive events depends on the dynamics of the system.

If we return to the example of the elevator, the events can be, for example, the arrival of the elevator at each floor. Following calls and stops, these events will not be equally distributed over time. An example of a discrete event system with a single state variable is shown in FIG. 1.4.

Considering the elevator again, we know that starting from the ground floor, it will reach the second floor only after passing through the first floor. However, the time required to reach this floor is not specified.

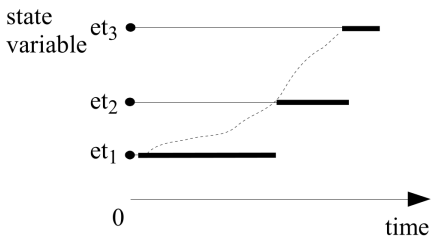


Figure 1.3: Discrete system

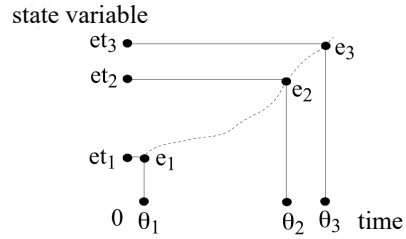


Figure 1.4: Discrete event system

Hybrid systems: Hybrid systems include both continuous state variables and discrete state variables. Dense time, coded by real numbers, and discrete time, in the form of a set of events, can both be used. This class of systems encompasses the previous four.

Discrete systems are typically contrasted with continuous systems. This classification depends on the observer's perspective and the desired level of abstraction. For example, consider a milling machine in a manufacturing system. From the perspective of milling operation, the system should be modeled as a continuous model. From the perspective of coordinating the manufacturing system, considering events such as *start of milling* and *end of milling*, the system should be modeled by a discrete event model.

1.2 Basic Notions

The Petri net is, originally, a tool for modeling discrete event systems. Let us detail the main concepts which are used.

1.2.1 Concepts Used in Modeling

The basic concepts used in modeling a system based on a discrete event approach are as follows:

Events: Events are the instants of observation and state change of the system.

Activities: Activities are black boxes used to encapsulate and hide the evolution of the physical system between two events. Therefore, events generally correspond to the start and end of an activity.

Processes: Processes are *sequences* of interdependent events and activities. For example, an event *triggers* an activity, which *leads to* an end-of-activity event, which in turn may *trigger* another activity, and so forth.

1.2.2 Parallelism, Cooperation, Competition

The evolution of processes in a system can occur simultaneously or not. If it occurs simultaneously, processes can be completely independent or relatively independent.

This relative independence means that certain activities are entirely independent of each other, while others require points of *synchronization*, that is, common events to several evolutions. There are different forms of interaction between processes:

Cooperation: Processes cooperate towards a common goal; the aim is to describe independence among processes *before* a synchronization point.

Competition: Processes must have access to a given resource to carry out their task. If there were a sufficient number of resources, the processes would be completely independent. Therefore, it involves resource sharing typically resolved by mutual exclusions. The aim is to describe exclusion between two processes *from* a synchronization point.

Pseudo-parallelism: Pseudo-parallelism occurs when parallelism is only apparent, and even though events are independent, they will never be simultaneous. They will be ordered by a common clock. This situation arises when multiple computing tasks are carried out on a single processor, which can only execute one instruction at a time.

True Parallelism: Events can occur simultaneously. This means that there is no sufficiently precise common time scale to determine which event preceded the other. It occurs when multiple computing tasks are executed on a parallel computer, with one processor allocated for each independent task.

1.3 Finite State Machine

1.3.1 Unique Sequential Process

The classic representation of a discrete-event system, where the number of states is finite, involves listing all possible *states*, and for each state, all the *events* that can occur and

which is the next state from this current state.

The mathematical model $M = (S, A, \theta, S_0)$ is called a *finite state machine*. It consists of a finite set of states S with an initial state S_0, an input alphabet A, and a state transition function $\theta : S \times A \to S$, which associates each state-input pair with the next state. A graph is associated with the model, where the set of nodes is the set of states S, and the arc leading from a state S_i to a state S_j is labeled by the event $a \in A$, such that $\theta(S_i, a) = S_j$.

This mathematical model effectively expresses the notion of an event and partially that of an activity (a state between two events); however, it does not express the notion of a process (simultaneous evolution of several parallel processes). A finite state machine, in fact, only describes a single sequential process.

Consider a simple sorting system on a moving conveyor belt C_b. Heavy objects need to be removed by an operator. A sensor P indicates the presence of a heavy object; when this happens, the conveyor belt must stop so the operator can remove the object. Once the object is removed, the conveyor belt resumes moving (FIG. 1.5.a).

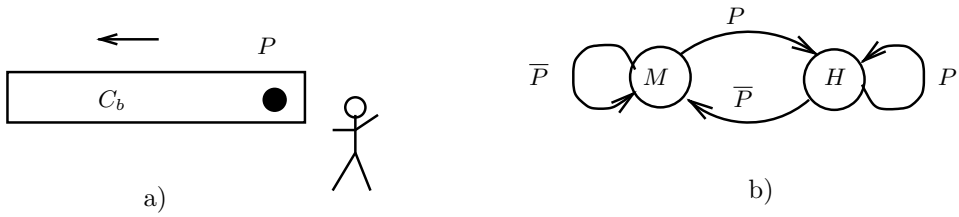


Figure 1.5: a) Sorting system; b) State Machine.

FIG. 1.5.b shows the state machine of this system: $S = \{M, H\}$ is the set of states. The initial state M represents the conveyor belt in motion, and the state H represents the conveyor belt halted. The alphabet $A = \{P, \overline{P}\}$ consists of the possible events that can occur in the system. The event P indicates the presence of a heavy object, and the event $\overline{P}$ indicates its absence. If the current state of the system is M, the occurrence of the event P leads the system to the state H. This behavior is represented by the state transition $\theta(M, P) = H$. The remaining state transitions are given by: $\theta(H, \overline{P}) = M$, $\theta(H, P) = H$, $\theta(M, \overline{P}) = M$.

1.3.2 Multiple Sequential Processes

When it is necessary to describe multiple sequential processes, the simplest solution is to represent the system by a set of finite state machines. If these machines are independent, there is no problem. However, when there is cooperation between various processes, the machines are not independent and must synchronize with each other.

Combinatorial Explosion of States

If there are shared information or signal exchange between the machines, it is necessary to analyze the global behavior of the system. This analysis can only be done by recalculating a state machine that describes the global system. In this case, the inevitable *combinatorial explosion* of the number of states occurs. Indeed, the composition of k

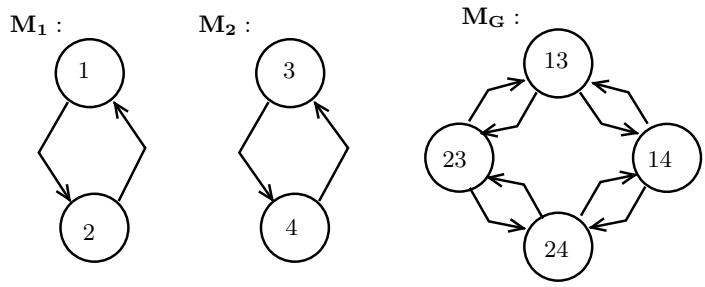


Figure 1.6: Combinatorial explosion of the number of states.

machines, in the particular case where each one has n states, produces a machine with n^k states.

In FIG. 1.6, both machines M_1 and M_2 have two states ($n = 2$) and the composed machine M_G has four states. Note in Table 1.1 that as n and k increase, the combinatorial explosion occurs:

n	2	2	2	3	3	4	4	4	6	6	10	6	6
k	2	3	4	3	4	2	3	4	2	3	3	4	5
n^k	4	8	16	27	81	16	64	256	36	216	10^3	1296	7776

Table 1.1: Combinatorial explosion n^k.

Non-independence of Submachines, Deadlock

Now, the analysis of the global behavior of the system is indispensable, as a number of serious problems can arise in parallel systems. The most well-known is deadlock. In certain configurations, no state machine can evolve because each machine awaits a particular evolution of another machine. All the machines are in waiting states. This phenomenon will be illustrated through an example in the following section.

1.3.3 Modeling Requirements

There are techniques in the theory of finite state machines that allow avoiding deadlock. However, an important point is that the structure of the system is completely lost. Two arcs (transitions) leaving the same state can represent a decision between two different options or two independent events. It should be emphasized that the main purpose of a clear specification of a discrete event system is to explicitly describe the interactions between the process states and the decision-making system that will control it. Another important point is that the introduction of even small modifications implies the construction of a new state machine.

The Petri net model that we will introduce next offers, in addition to behavioral knowledge about the system, also structural knowledge as will be seen throughout this book.

1.4 Informal Presentation of a Petri Net

1.4.1 Basic Elements

A Petri net is a directed graph having two types of nodes: *places*, represented by circles and *transitions*, represented by rectangles or just a line. The arcs of the graph can only connect places to transitions, or transitions to places (never between places or between transitions).

A Petri net describes a dynamic discrete event system. Places allow the description of the states (which are therefore discrete) and transitions allow the description of events (state changes). In general, every place has a predicate associated with. Consider the Petri net in FIG. 1.7: transition t_1 is associated with the event *start operation* while places p_1 to p_3 are labeled to indicate the associated predicate.

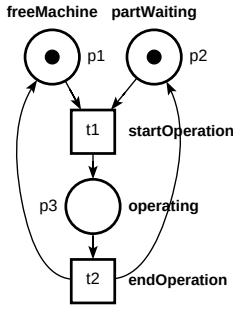


Figure 1.7: A Petri net model.

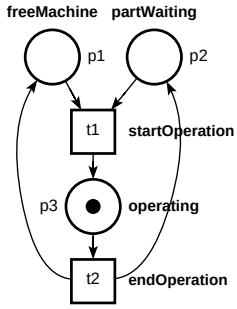


Figure 1.8: After t_1 firing.

A Petri net is a graph with an operational semantics, that is to say that a behaviour is associated with the graph, which makes it possible to describe the dynamics of the represented system. For this, a third element is added to places and transitions: the *token*, represented by a dot inside a place.

A distribution of tokens over the places at a given time is called a *marking* of the Petri net. A marking gives the state of the system. The number of tokens contained in a place is a natural number (non-negative integer). The Petri net in FIG. 1.7 depicts the state where there is one *free machine* (one token in p_1) and one *part waiting* (one token in p_2). As there is no token in place p_3, the predicate is false, indicating that there no machine *operating*. If there were three tokens in place p_2, this would indicate that there are three parts waiting.

The first observation to make is that interpretations of places and tokens are quite varied. They can be used to describe abstract entities such as conditions or states, but also physical entities such as parts or machines. One can also achive a complete simplification, at the descriptive level, between the parts, tools, and, in general, between all the resources used in the factory. It is this great simplification that allows for a synthetic view of the system to be modeled and permit certain analysis procedures.

1.4.2 Dynamic behaviour

The system's state is given by the distribution of tokens in the places of the Petri net, with each place representing a partial state of the system. Each event that occurs in the system is associated with a transition in the Petri net model. The *occurrence* of an

event in the system (which causes it to move from the current state to the next state) is represented in the model by the firing of the transition to which it is associated.

The firing of a transition consists of two steps:

- Remove a token in each of its input places, indicating that this condition is no longer true after the event occurrence, and
- Add a token to each of its output places[1], indicating that these activities will, after the event occurrence, be performed.

A transition can fire only if it is *enabled*. A transition is enabled if each of its input places contains at least one token[2]. The set of all the transitions enabled by a given marking defines the set of possible state changes for the system from the state corresponding to this marking. It is a way to define the set of events to which this system is *receptive* in this state.

Consider again the Petri net in FIG. 1.7: transition t_1 is enabled and can be fired, leading to the marking represented in FIG. 1.8.

For example, the occurrence of the event *start operation*, associated with the transition t_1 (FIG.1.7), can only happen if there is (at least) one token in the place *freeMachine* and (at least) one token in the place *partWaiting*. The occurrence of the event *start operation* in the system is equivalent to the firing of the transition t_1 in the Petri net: a token is removed from places *freeMachine* and *partWaiting*, and a token is placed in the place *operating* (FIG.1.8).

The firing of a transition t corresponds to the occurrence of an event e in the real system, which causes it to move from its current state S_i to the next state S_{i+1} (see Section 1.3). The state S_i is represented in the net by the distribution of tokens in the places, called marking M_i. Just as the system will only reach state S_{i+1} after the occurrence of event e if it is in state S_i, the transition t will only be fired if the marking is M_i (a marking in which, in particular, the input places of t are marked). The marking M_{i+1}, corresponding to state S_{i+1}, will be reached after the firing of transition t.

The removal of tokens in the input places of t indicates that the conditions or predicates associated with those places are no longer true, and the addition of tokens in the output places indicates that the predicates associated with these places become true. Thus, the dynamic behavior of the system is translated by the behavior of the Petri net.

1.5 Modeling Interactions Between Processes

As seen in Section 1.2.1, processes can evolve in cooperation, competition, and in parallel. Depending on the level of detail in the modeling, a process can be represented by only one place or activity. Processes can also evolve sequentially, repeatedly, etc. In the following sections, it will be shown how to model, using Petri nets, the different ways processes interact in discrete event systems.

[1] The weights associated with the arcs allow you to remove or add several tokens to a given place.

[2] We will see during the formal definition that weights can be associated with the arcs which imposes then the presence of several tokens in a given place.

1.5.1 Sequence

Consider the Petri net in FIG. 1.9. This same model can represent:

- Sequence of a manufacturing process. Places P_1, P_2, and P_3 represent the different phases of the operation on the part, which must be performed in sequence. Transitions t_1, t_2, and t_3 describe the events of *passing from one phase to another*, and the tokens correspond to the parts. A part is being machined at P_1 (phase J_1), while another part is currently in phase J_2, having already passed through phase J_1 (places P_1 and P_2).

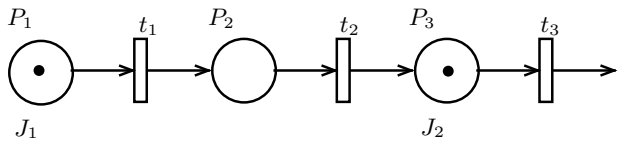


Figure 1.9: Sequence of processes.

- A section of the itinerary of a transportation system. This system is based on vehicles that follow pre-drawn circuits on the ground. In this case, places describe the sections; transitions correspond to the passage of the vehicle from one section to another (passing by a sensor on the ground), and the tokens represent the vehicles. A vehicle in section S_i is represented in the Petri net by a token in place P_i. The event *leave section* S_i *and enter section* S_{i+1} is associated with transition t_i. Thus, the marking of the net in FIG. 1.9 represents one vehicle in P_1 and another vehicle in P_3, indicating that it is possible for the events *leave* S_1 and *leave* S_3 to occur in parallel.

In both cases, the Petri net is the same, which expresses that, by abstracting from the details and considering only the structure of the chains of events and activities, these two systems are identical.

1.5.2 Synchronous and Asynchronous Evolutions

If the Petri net in FIG. 1.9 represents the model of a manufacturing process, the tokens in P_1 and P_3 indicate that simultaneously, there is one item in phase J_1 and another item in phase J_2. Similarly, if this figure represents a segment of a transportation system, then it can be seen that simultaneously two vehicles are traveling on the same route, in different sections (P_1 and P_3).

The evolution of these two tokens (parts or vehicles), as described by the Petri net, is independent and evolves in a totally asynchronous manner: there is no correlation between the end of phase J_1 and the end of phase J_2.

In the case of FIG.1.10.a, a division or separation procedure is described. The end of operation P_1 consists of separating an item (in this case J_1) to create two new items corresponding to different manufacturing processes. Therefore, the two items J_2 and J_3 are created, simultaneously, in a synchronous manner.

From this point of view, they are not independent. However, after their creation, they evolve independently of each other in an asynchronous manner.

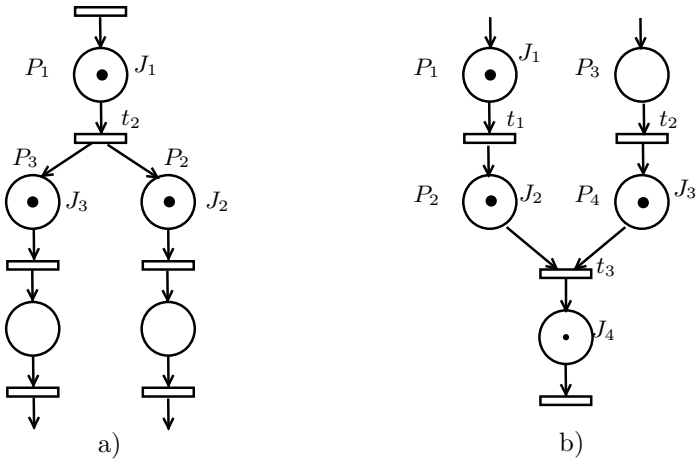


Figure 1.10: a) Fork; b) Joint.

The event removal of an item J_1 and the addition of two more items J_2 and J_3 is clearly and succinctly described by transition t_2. The marking in FIG.1.10.a indicates that the item J_1 is undergoing operation P_1 (a token in P_1) and that, at the same time, the items J_2 and J_3 are also in operation (places P_2 and P_3 marked).

In the case of FIG. 1.10.b, the items evolve independently, in parallel, and asynchronously (transitions t_1 and t_2) in different processes, except for transition t_3, which requires the simultaneous presence of a token in P_2 and a token in P_4 to be fired. This transition can be associated, for example, with the event *start assembly*. In this case, places P_2 and P_4 might represent parts waiting in storage areas.

The removal of these two tokens (upon the firing of t_3) will be synchronous and will simultaneously create a new token that will undergo the first operation of its manufacturing process (J_4). The transition from these asynchronous evolutions to a synchronous evolution necessarily implies a wait in places P_2 or P_4, depending on which condition is met first.

The Petri net in FIG. 1.11 is an example of a "fork join" structure. Transitions t_1 (fork) and t_4 (join) allow for independent firing of transitions t_2 and t_3.

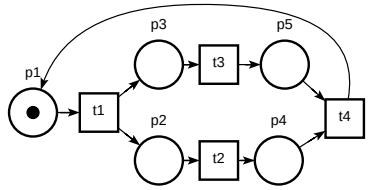


Figure 1.11: Parallel evolution of t_2 and t_3.

1.5.3 Variants and Alternative Paths

FIG. 1.12.a represents the case where, after a phase (or operation) P_1, there is a choice between the sequences P_2–P_3 or P_4–P_5. Then, P_6 is executed. To execute the sequence

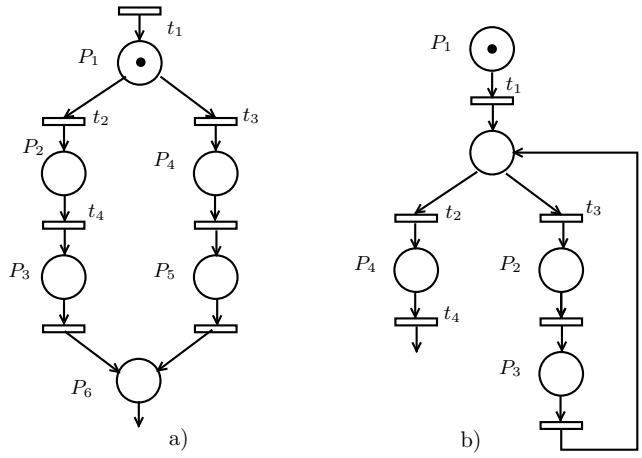


Figure 1.12: a) Alternative paths; b) Repetition.

P_2–P_3, transition t_2 must be fired, and to execute P_4–P_5, transition t_3 must be fired. Although both transitions are enabled, only one of them can be fired because the token is removed from place P_1 when the chosen transition is fired. From that moment, the other transition can no longer fire. However, the structure of the Petri net does not provide any information about the decision-making mechanism for choosing the alternative to be executed (t_2 or t_3). It merely indicates that this information must be available at the end of phase P_1.

If the Petri net models the segment of the itinerary of a transportation system — with place P_i associated with the traversal of section S_i — then place P_1 represents the vehicle in a section that allows for two alternative paths: passing through sections S_2 and S_3 *or* through sections S_4 and S_5 before continuing through section S_6.

By comparing FIG.1.10.a with FIG.1.12.a, the following observations can be made:

- A transition with more than one outgoing arc (t_2 in FIG. 1.10.a) corresponds to a fork or the start of parallel evolutions. A fork is typically followed by a joint as in FIG. 1.10.b.

- A place with more than one outgoing arc (P_1 in FIG. 1.12.a) corresponds to the start of a set of alternative paths.

1.5.4 Repetition

Another behavior that needs to be modeled when representing discrete event systems is the repetition of an activity (or sequence of activities) while a condition is true. For example, a vehicle should repeat passing through the circuit formed by sections S_2 and S_3 until an order to change the route comes or until it's necessary to recharge the battery.

Consider the Petri net in FIG. 1.12.b (once again, each place P_i is associated with the crossing of section S_i). This Petri net models the possibility of executing the sequence P_2-P_3 a certain number of times after P_1, before carrying out P_4. Once again, there is no indication at the graph level about the test that should be performed to decide between repetition or termination of the sequence. It is only known that the result should be

available after the end of operation P_3 (which could precisely be the execution of this test).

Consider again the example of the transportation system itinerary. The vehicle must go through sections S_2 and S_3 several times, and after go to S_4. This behavior can be modeled by the Petri net in FIG. 1.12.b.

The coordination level indicates which section (S_2 or S_4) the vehicle should follow, and this is not part of the net structure [3]. The net structure should only indicate the two existing possibilities for the vehicle.

1.5.5 Resource Allocation

Without a doubt, the utilization of resources, especially their sharing, is one of the most important aspects in modeling a system. A vehicle that must carry one of several parts at a given time, a robot that can transport a part from the depot to the machine and from the machine to the exit, can perform only one activity at a time. Once engaged in one task, it cannot be available for the another. Although the concept is trivial, ensuring it is not always straightforward. Therefore, modeling resource sharing is fundamental for the accurate representation of a system.

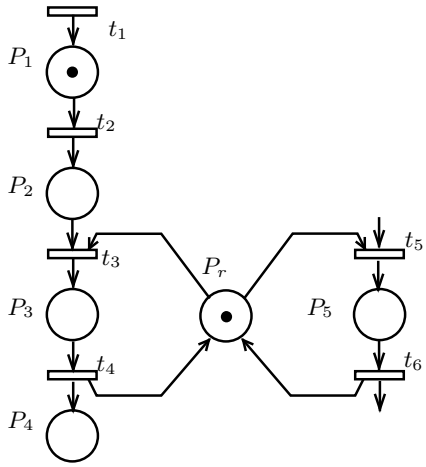


Figure 1.13: Resource sharing.

Observing FIG. 1.13, consider that after an activity P_1, it is necessary to perform an operation that requires the use of a resource r represented by the place P_r in the figure. A token in P_r corresponds to the state *resource available*. Transition t_3 expresses the acquisition of the resource and the beginning of phase P_3.

To model separately the *end of activity* P_1 and the *beginning of* P_3, it is necessary to introduce transition t_2 associated with the end of activity P_1. Place P_2 corresponds to waiting for the resource associated with place P_r if it is not available (this place plays a similar role to places P_2 and P_4 in the Petri net of FIG. 1.10.b).

The absence of t_2 and P_2 in this Petri net would model a different behavior. Since t_3 can only fire if place P_r is marked, waiting for the resource would, in this case, block the end of activity P_1.

[3] This type of information is part of the *interpretation* and will be discussed in Chapter 4.

Each use of the resource corresponds to the execution of a loop, which begins with *acquiring* of the resource (firing of t_3 in FIG. 1.13) and ends with its *release* (firing of t_4 in FIG. 1.13). There are as many different loops passing through place P_r as there are possible uses of the resource.

The resource r is treated exactly like a part. Thus, the concepts of part and resource, represented by tokens, are banalized. If the resource r is part of a set of resources, then place P_r directly represents this set. Initially, you just need to place as many tokens in this place as there are elementary resources.

Example 1.1 Transportation System
Consider the itinerary of a transportation system, based on automatically guided vehicles, as shown in FIG. 1.14.a. The vehicles follow predefined circuits automatically; their locations are only known at points C_i, called *contacts*. The commands to *stop*, *continue*, and *change route* are sent to the vehicles when they are on the contact.

To avoid collisions, the circuits are divided into sections in such a way that only one vehicle can be in each section. Therefore, before entering the next section of the circuit, it is necessary to check if it is free: the section is thus considered as a resource to be shared among the different vehicles. The section appears dashed in the figure and has the same name as the contact.

The circuits can have four types of sections: converging, diverging, intersecting, and sequential.

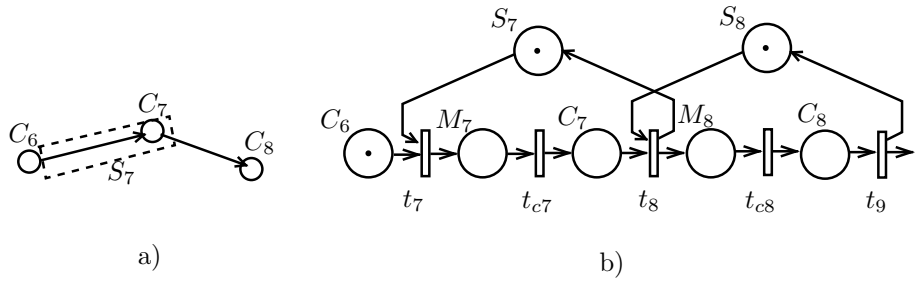


Figure 1.14: a) Sequential sections. b) Model of circuit $N_0 = \{C_6, S_7, S_8\}$.

Sequential sections
Consider the circuit N_0 shown in FIG. 1.14.a, formed by contact C_6 and sections S_7 and S_8, modeled by the Petri net in FIG. 1.14.b. Each proposition *section S_i free* corresponds to a place S_i. There are two predicates associated with each section: *vehicle in motion*, associated with place M_i, and *vehicle stopped at contact*, associated with place C_i. The event *vehicle enters the section S_{i+1}* is associated with transition t_{i+1}. Its occurrence happens when a vehicle is at the contact of section S_i (place C_i) and the next section S_{i+1} is free. After firing transition t_{i+1}, the vehicle releases section S_i, and is in motion in section S_{i+1} (place M_{i+1}). The event *vehicle stops at the contact C_i* is associated with transition t_{ci}. The initial state of the system has a vehicle at the contact of section S_6, with sections S_7 and S_8 free (places C_6, S_7, and S_8 are marked, respectively).

Closed circuit

Let us consider only the circuit N_1 shown in FIG. 1.15.a, formed by sections S_1, S_2, and S_3. This circuit is closed: after traversing section S_3, the vehicle returns to section S_1. The Petri net model of this circuit is shown in FIG. 1.15.b. The marking indicates that the vehicle is stopped at contact C_3, while sections S_1 and S_2 are free.

Converging/diverging sections

Let us now consider the two circuits, N_1 (S_1, S_2 and S_3) and N_2 (S_1, S_4 and S_5), in FIG. 1.15.a. When considered independently, circuit N_2 can also be modeled by a Petri net similar to the one in FIG. 1.15.b. However, circuits N_1 and N_2 share contact C_1 (contact at the end of section S_1), which acts a converging section when approaching from C_3 or C_5, and as a diverging section when going toward C_2 or C_4. The corresponding Petri model for the sections shown in FIG. 1.15.a is represented in FIG. 1.15.c. The marking indicates that there is a vehicle at C_3 and another one moving in section S_5.

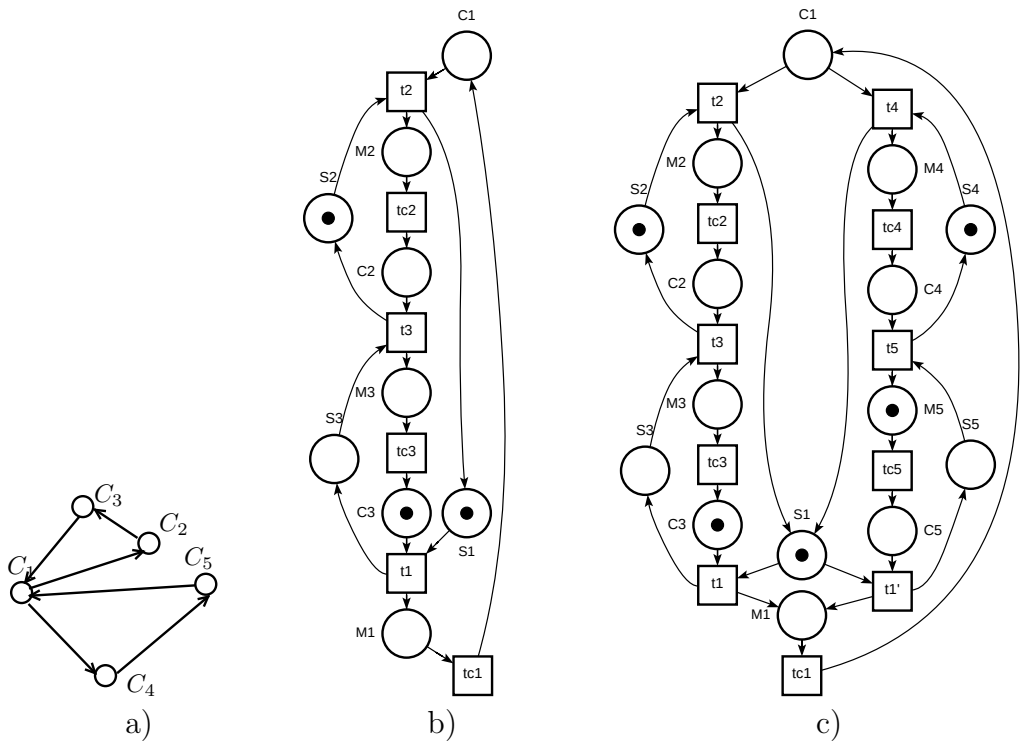


Figure 1.15: a) Converging/diverging sections; b) Circuit N1; c) Circuits N1 and N2.

Two markings are of particular interest in this model:

- A vehicle at C_1 and another one at C_5, with sections S_2, S_3, and S_4 free. The model shows that the vehicle at C_1 can follow either circuit N_1 or N_2, represented by transitions t_2 and t_4, which are in conflict.
- A vehicle at C_3 and another one at C_5, with sections S_1, S_3, and S_4 free: The model indicates that only one of them can enter section S_1 and move toward C_1, represented by transitions t_1 and t'_1 in conflict.

Intersecting sections

Let us now consider the circuits represented in FIG. 1.16.a, where there are intersecting sections. In this case, besides the notion of a free section, it is necessary to introduce the notion of a free cell. Therefore, in addition to checking if the next section S_{i+1} of the circuit is free, it is necessary to check if the other section S' of the cell is also free. Otherwise, during the crossing of section S_{i+1}, the vehicle could collide with another vehicle crossing section S' in the same cell.

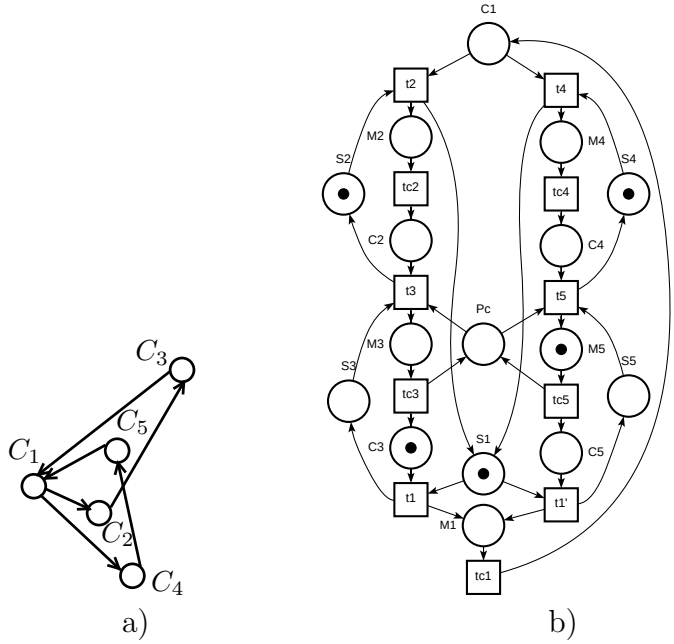


Figure 1.16: a) Intersecting sections. b) Petri net model of circuits N_1 and N_2.

In the circuits of FIG. 1.16.a, the intersection cell, formed by sections S_3 and S_5, must be considered as a resource. Before entering a section of this cell, it must be checked if it is free, that is, if there is no moving vehicle in S_3 and S_5. However, a vehicle can be stopped at the contact of the other section, as there will be no risk of collision in this case. For this reason, the cell is released when the vehicle arrives at the contact.

The FIG. 1.16.b represents the complete model of circuits N_1 (S_1, S_2 and S_3) and N_2 (S_1, S_4 and S_5) of the transportation system, where the place P_c corresponds to the intersection cell. The marking of the Petri net represents the system state where a vehicle is crossing section S_5, and, as a result, the intersection cell is no longer free (no token in P_c). ◇

Example 1.2 Batch-type System

A batch-type system can produce two products (Pr_1 and Pr_2) using two reactors in concurrent mode, as depicted in FIG. 1.17.a. Product Pr_1 can be produced by reactor R_1 or reactor R_2, and must be stored beforehand in buffer B_1 or B_2, respectively. Product Pr_2, on the other hand, can only be produced by reactor R_2, where it is directly loaded. The system's behavior is repetitive: once the product is ready, each reactor is released and can start a new activity. Although reactor R_2 can handle two types of products, it does so one at a time. The model of the system's behavior is given by the Petri net

in FIG.1.17.b. Reactors R_1 and R_2, represented by places p_8 and p_9, respectively, are considered as resources, with R_2 shared between the two batches of products Pr_1 and Pr_2.

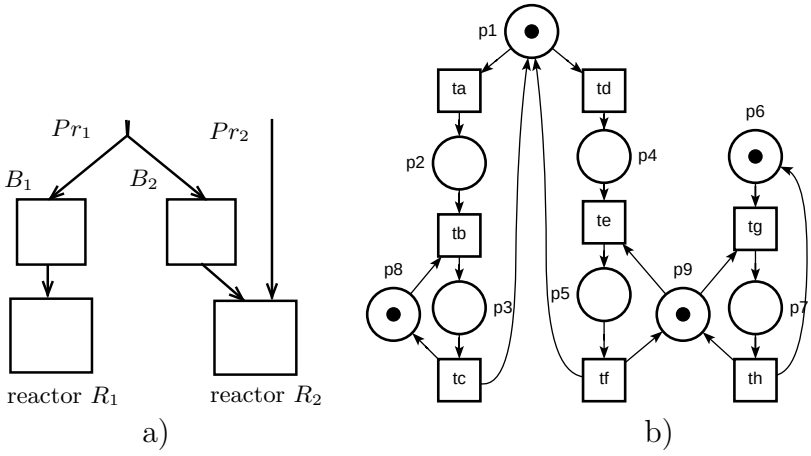


Figure 1.17: a) Batch system; b) Petri net model.

Note that when place p_1 is marked, indicating that there is a batch of product Pr_1 to be produced, there are two alternative paths, represented by the transitions t_a and t_d, associated, respectively, with the events *store in* B_1 and *store in* B_2, since the product Pr_1 (place p_1) can be stored in both buffers. Place p_2 (p_4) represents the activity *product* Pr_1 *stored in buffer* B_1 *(*B_2*)*, to be processed later by reactor R_1 (place p_8) or reactor R_2 (place p_9), respectively. Transition t_b (t_e) is associated with the event *start operation of reactor* R_1 *(*R_2*) on the batch of product* Pr_1, respectively.

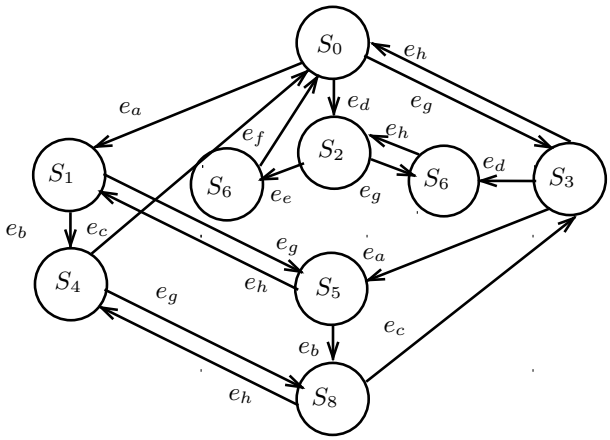
The transition t_g is associated with the event *start of operation of reactor* R_2 *on the batch of product* Pr_2 and indicates the occupation of resource R_2 (place p_9), while transition t_h indicates its release (arc to place p_9) and permission for a new batch of Pr_2 to be processed. Transitions t_e and t_f also indicate occupation and release of R_2, but in relation to product Pr_1.

From the perspective of place p_9, there are also two alternative paths: process product Pr_1 (firing t_e) or process product Pr_2 (firing t_g). The structure of the net models only the possible behaviors for the system without indicating which choice will be made.

This system can also be modeled by a global finite state machine, represented in FIG. 1.18. The nodes S_i, $i = 0 \ldots 8$, represent the different states of the system, and the arcs indicate, for a state S_i, the next state S_{i+1}. The events $e_a, \cdots, e_h$, labeling the arcs, are the same events associated with the transitions $t_a, \cdots, t_h$ of the Petri net model.

Note, as discussed in Section 1.3.3, that the structure of the system has been completely lost. Two arcs leaving the same node (state) can represent a decision between two options or two independent events. For example, the arcs from node S_1 are labeled by e_b (allocation of R_1 by Pr_1) and e_g (allocation of R_2 by Pr_2), these being independent events that can occur simultaneously. On the other hand, the events e_a and e_d, labeling the arcs leaving state S_0, cannot occur simultaneously and are concurrent.

The choice between the two manufacturing options for product Pr_1 is clearly indicated in the Petri net model of FIG. 1.17.b: place p_1 has two outgoing transitions (t_a and t_d),



$S_0 : Pr_1$ and Pr_2 waiting, reactors free
$S_1 : Pr_1$ in B_1, Pr_2 waiting, reactors free
$S_2 : Pr_1$ in B_2, Pr_2 waiting, reactors free
$S_3 : Pr_1$ waiting, R_2 processing Pr_2, R_1 free
$S_4 : R_1$ processing Pr_1, Pr_2 waiting, R_2 free
$S_5 : Pr_1$ in B_1, R_2 processing Pr_2, R_1 free
$S_6 : R_2$ processing Pr_1, Pr_2 waiting, R_1 free
$S_7 : Pr_1$ in B_2, R_2 processing Pr_2, R_1 free
$S_8 : R_1$ processing Pr_1, R_2 processing Pr_2

Figure 1.18: Finite state machine of a batch system.

indicating the two alternatives (storage in B_1 or B_2). The same goes for reactor R_2, which can process Pr_1 or Pr_2, represented by the alternatives t_e and t_g. In both cases, these transitions are in competition. However, transitions t_b and t_g, when places p_2, p_6, p_8, and p_9 are marked, can be fired independently. Therefore, the Petri net indicates, within its structure, when there is parallelism or concurrency between events. It is interesting to revisit this example after reading Chapter 2, where the notions of conflict and parallelism are formally defined. ◇

1.6 How to Start a Petri Model Design

In this section we will see how to model, from a textual specification, the dynamic behaviour of a system using ordinary (or classical) Petri nets (PN), showing sequences, parallelism (or independence) and conflict between processes.

You can use the Tina[4] toolbox for the to assist you with edition, simulation and analysis. Let us start by solving the Exercise 4 proposed in Section 1.8.

1.6.1 Determining activities and events

How to describe a discrete event system in terms of activities and events? It's not always easy to start from scratch. Events are typically named with verbs in the imperative form, signifying a specific moment of change or observation, and are associated with transitions in the Petri net. Activities, on the other hand, are named with verbal noun (ending in *ing*) or past tense verbs (ending in *ed*), indicating ongoing or finished processes, actions or states. They are associated with places in the Petri net.

You can start for checking if there are resources, and write them down. After that, you can consider them as they are conditions (pre- and post-conditions of some events). Notice that modelling is a process that goes back and forth.

Then, for helping the brainstorming, create a table of events relating them with the input and output activities (or conditions, or predicates). In other words, indicate the

[4]TINA has been developed at LAAS, see more information in Appendix E and here: https://projects.laas.fr/tina/index.php

(pre)conditions that enable an event to occur in the system, and which partial states are modified (post-conditions). Remember that an activity is related with a state variable, and represents a partial state of the system.

Consider the system described in Exercise 4 in Section 1.8. Table 1.2 describes the event *take a part from input buffer B_i (to put on lathe L)*. While filling the table, you can start to draw your Petri net as in FIG. 1.19: activities are represented by places, and events by transitions. This can help you to find other activities and events that you did not see in the beginning.

Event	Pre-condition	Post-condition
takePartFromB_i2L	R is free Part in Bi	movingPartB_i2L
...	...	...

Table 1.2: Table of events.

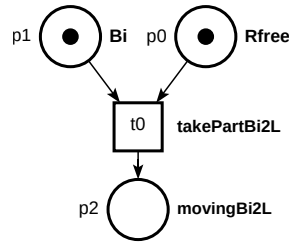


Figure 1.19: Petri Net model.

You can also create a table of activities (and resources) relating them with the input and output events. Table 1.3 describes the activity *moving a part from the input buffer B_i to lathe L*. Sometimes it also helps to find missing elements (places or transitions). The corresponding Petri net from Table 1.3 is shown in FIG. 1.20.

Activity	Input Event	Output Event
movingPartB_i2L	takePartFromB_i2L	putPartOnLathe
...	...	...

Table 1.3: Table of activities.

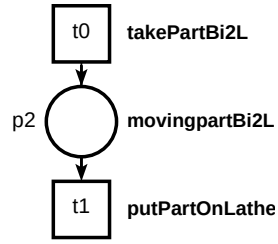


Figure 1.20: Petri Net model.

Appendix C describes the modeling process of this example, including the verification and validation of the Petri net model, the theory of which is presented in Section 3.

1.6.2 Simulation

The simulation may be used on a partial Petri net during the design stage to verify if the interaction between the processes (parallelism, conflict, sequence) corresponds to the specifications. The simulation can then be used to either correct the model (e.g., simulating a sequence leading to a deadlock) or to observe its behavior (using transition invariants, which will be presented in Section 2.5.2).

When doing the simulation, in your paper or using Tina, write scenarios in English and *translate* them to the corresponding firing of transitions. An example of a scenario in English is to show, for Exercise 4, the "life cycle" of a part as it goes from the input buffer to its exit from the shopfloor.

However, analysis is required for validation and verification. These will be seen in Chapter 3.

1.7 Notes

The concept of discrete event systems is presented in CASSANDRAS (1993). For an introduction to finite state machine theory, see ROSEN (1991) and JOHNSONBAUGH (1993). For a comprehensive study, see CARROL & LONG (1989). In the area of concurrent systems, ANDREWS (1987) provides a good introduction for those unfamiliar with the subject.

Additional reading for the entire Part I, which deals with the basic model: PETERSON (1981), BRAMS (1983), COURVOISIER & VALETTE (1986) and DAVID (1989).

1.8 Exercises

The exercises below are proposed for reflection. If you encounter difficulties at this moment, leave them for after reading Chapter 2.

Suggestion: Before drawing the Petri net, as introduced in Section 1.6, detail the following:

- The activities and events present in the system;
- The resources to be shared;
- Which activities can be performed in parallel and which are performed concurrently.

1 A manufacturing system has two machines M_1 and M_2. Two types of manufacturing processes are planned for a part, according to the order of passage through the machines: $P_1 = O_{11}O_{12}$ and $P_2 = O_{21}O_{22}$, where O_{ij} is the j-*th* operation of the manufacturing process P_i. Machine M_1 performs operations O_{11} and O_{22}, and machine M_2 performs operations O_{12} and O_{21}. Model this system using Petri net, considering two different cases: a) before performing the second operation of its manufacturing process, the machine from the first operation must be released; b) the machine from the first operation is released only after the start of the second operation of its manufacturing process.

2 In a small town with a fire hydrant on each street, there are two firefighters. When a fire occurs, one of the firefighters stands next to the hydrant, and the other next to the fire. The firefighter next to the hydrant fills the bucket and goes towards the fire; the firefighter next to the fire, once the bucket is emptied, goes towards the hydrant. When they both meet, they exchange buckets and directions (the one coming from the hydrant goes back to fill the empty bucket of the colleague; the one coming from the fire returns with the full bucket that the colleague brought). Consider that, even if the firefighter who fills the bucket is: i) very fast, he will not arrive at the fire before the other empties his; ii) very slow, he will have filled the bucket before his colleague reaches the hydrant with the empty bucket.
a) Model, using Petri net, the behavior of the two firefighters;
b) Consider that a third firefighter comes to help, standing in the middle of the way to minimize the displacement of the other two. Make the model of the new system.

3 Model using Petri nets the structure of the speed control system of a motor using thyristors. Initially, the control system must be initialized (initialize variables, analog-to-digital and digital-to-analog converters, etc.). The control cycle consists of taking temperature and flow measurements (independently) and then calculating according to the control law. In the particular case of temperature measurement, if it exceeds a limit L_T, an alarm must be triggered. At the same time as the firing angle is calculated and sent to the digital-to-analog converter, a calculation for later statistics is performed. Once the calculations are completed, the cycle starts again.

4 Design, using a Petri net, a manufacturing cell composed of a robot R, a lathe L, and a laser micrometer M (FIG. 1.21), interconnected by a communication network, which operates as follows:

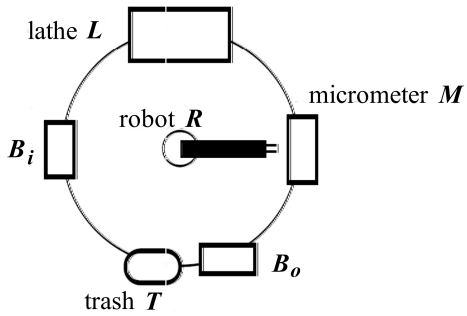


Figure 1.21: Shopfloor.

- The parts are deposited in the *input buffer* B_i;
- The robot picks up the part from B_i and places it in the lathe L (if L is free). A program must be loaded to machine the part;
- After machining, the robot picks up the part from the lathe T and places it in the laser micrometer M (if it is free) for being mesured;
- After inspection, the robot R places the part in the *output buffer* B_o if it meets the standards, or in the trash T if it does not. The capacity of the trash and B_o are infinite.
- The robot can do only one operation at once; the lathe L and the micrometer M can only work on a part at a time and they have no buffer.
- Each time a part is removed from the input buffer, a new one is introduced.
- The operation duration on L is longer than on M.

a) Consider that there is only one type of part, and that the *input buffer* B_i has the capacity to store only one part.
b) Consider now that the capacity of B_i is two parts.
c) How would the Petri net look if there were two types of parts?

You can find the solution of this exercise in Appendix C.

Chapter 2

Definitions

This chapter presents Petri nets as a formal model through three perspectives:

- a graph with two types of nodes and dynamic behavior;
- a set of matrices of non-negative integers whose dynamic behavior is described by a linear system;
- a rule-based system represented as *condition* $\rightarrow$ *action*.

The concepts of marking and transition firing, introduced informally in the previous chapter, are formally defined here for each of the above viewpoints.

The graphical and matrix viewpoints, presented in Section 2.1, differ only in their representation form. Both allow us to check if transitions are parallel or in conflict, if a transition is enabled, as well as fire a transition and evolve the net. The graphical representation is used by the designer, who can quickly make all these checks and have a global view of the modeled system (of reasonable size). The matrix representation is a natural format for the computer, which can perform automatic verification for very large models.

The notion of a Petri net simulator, or token-player, is presented in Section 2.2.

The representation using a rule-based system aims to make the Petri net representation compatible with Artificial Intelligence techniques and is presented in Section 2.3. In the specification of a complex system using a multi-level control hierarchy, the modeling of the subsystem coordination level can be done with Petri nets, and the planning level with a knowledge-based system that provides the scheduling of the parts on the machines.

Although the graphical representation is an advantage of Petri nets, as the reader has already had the opportunity to observe, the most important characteristic of this model is the fact that it is formal. The advantage of Petri nets over other models, which also offer good graphical specification tools, is that, being formal, they allow for the extraction of information about the behavior of the modeled system through the analysis of its general or structural properties.

The general properties are presented in Section 2.4. The conservative and repetitive components, which allow the analysis of structural properties, are defined in Section 2.5. The methods for analyzing these properties are presented and discussed later in chapter 3.

2.1 Concepts

2.1.1 Petri nets

A Petri net is a quadruple

$$R =< P, T, Pre, Post > \tag{2.1}$$

where:

- P is a finite set of places of dimension n;
- T is a finite set of transitions of dimension m;
- $Pre : P \times T \rightarrow I\!N$ is the *input* function (preceding places or pre-incidence), with $I\!N$ being the set of natural numbers;
- $Post \times T \rightarrow I\!N$ is the *output* function (following places or post-incidence).

The quadruple $R =< P, T, Pre, Post >$ with $P = \{p_1, p_2, p_3\}$, $T = \{a, b, c, d\}$ and the input and output function values given by $Pre(p_2, c) = 3$, $Pre(p_1, b) = Pre(p_2, a) = Pre(p_3, d) = 1$, $Post(p_2, d) = 3$ and $Post(p_1, a) = Post(p_2, b) = Post(p_3, c) = 1$, is a Petri net.

2.1.2 Marked Petri Net

A marked Petri net N is a pair

$$N =< R, M > \tag{2.2}$$

where:

- R is a Petri net,
- M is the initial marking given by the application

$$M : P \rightarrow I\!N. \tag{2.3}$$

$M(p)$ is the number of tokens contained in place p. The marking M is the distribution of tokens in the places, represented by a column vector whose dimension is the number of places and components $M(p)$.

The pair $N =< R, M >$ with R being the Petri net described in Section 2.1.1 and marking $M^T = [\ 0\ 3\ 0\ ]$ (T is the transpose of the vector) is a marked Petri net.

Notation:
For the sake of readability, a marking can also be represented by listing the places p_i^j that have $M(p_i) = j$, where $j > 0$. The marking $M^T = [\ 0\ 3\ 0\ ]$ can then be written as $M = p_2^3$. Both notations will be used in this book.

2.1.3 Associated graph and matrix notation

A Petri net can be associated with a graph with two types of nodes: place nodes and transition nodes. An arc connects a place p to a transition t if and only if $Pre(p,t) \neq 0$. An arc connects a transition t to a place p if and only if $Post(p,t) \neq 0$.

From the elements $a_{ij} = Pre(p_i, t_j)$, which indicate the weight of the arc connecting input place p_i to transition t_j, the pre-incidence matrix Pre of dimension $n \times m$ is defined: the number of rows is equal to the number of places and the number of columns is equal to the number of transitions. Similarly, the post-incidence matrix $Post$ of dimension $n \times m$ is defined from the elements $b_{ij} = Post(p_i, t_j)$.

The non-zero values of the matrices Pre and $Post$ are associated with the arcs of the graph as labels. If the value is unity, it is not necessary to label the corresponding arc in the graph. Similarly, if nothing is indicated in the graph, the corresponding value in the matrix is assumed to be unity.

The notation $Pre(\,.\,,t)$ is used for the column of matrix Pre associated with a transition t. The dimension of this vector is given by the number of places. Similarly, the column vectors $Post(\,.\,,t)$ and $C(\,.\,,t)$ are defined with respect to the matrices $Post$ and C, respectively.

Example 2.1 Matrix notation
The associated graph with the marked Petri net N from Section 2.1.2 is depicted in FIG. 2.1. This net represents the sharing of a set of resources (three), represented by place p_2, between two activities, represented by places p_1 and p_3. The activity corresponding to p_1 requires only one resource at a time (the weight of the arc (p_2, a) is 1, or $Pre(p_2, a) = 1$).

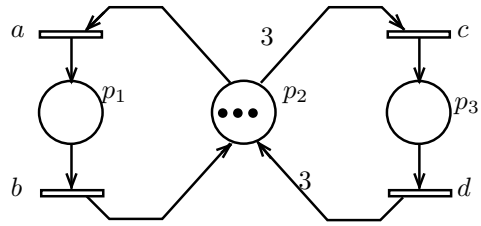


Figure 2.1: Resource sharing.

On the other hand, the activity corresponding to p_3 requires all three resources simultaneously. Also note that there is a mutual exclusion between p_1 and p_3: after firing c, transition a cannot fire anymore, and vice versa. However, after firing a, it can still fire two more times. Therefore, there can be three p_1 activities being executed simultaneously.

This model encompasses the classical problem in computer science of readers and writers, where multiple readers (place p_1) can access a resource (p_2) simultaneously, but a writer (p_3) requires exclusive access to the resource.

The matrix notation of this Petri net is given by:

$$Pre = \begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix} \qquad Post = \begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}.$$

From Pre and $Post$, the incidence matrix C is defined as

$$C = Post - Pre \tag{2.4}$$

which provides the *update* of tokens in the net when firing the transitions.

In this example, we have: $C = \begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2. \\ p_3 \end{matrix}$

◇

2.1.4 Pure Petri Net

A Petri net is pure if and only if

$$\forall p \in P \ \ \forall t \in T \ Pre(p,t)Post(p,t) = 0. \tag{2.5}$$

The equivalent graph has no *elementary loop*, that is, no transition has the same place as both input and output at the same time.

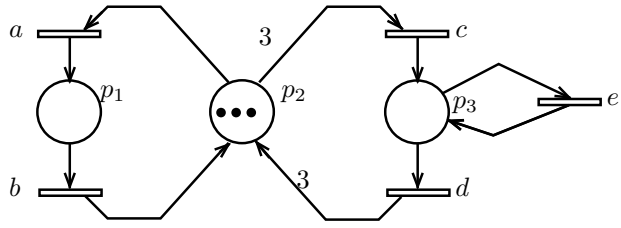


Figure 2.2: Non-pure Petri net.

For example, the Petri net in FIG. 2.1 is pure. However, the net in FIG. 2.2 is not pure, as place p_3 serves as both input and output for transition e:
$Pre(p_3, e)Post(p_3, e) = 1$.

2.1.5 Enabled Transition

A transition is enabled if and only if:

$$\forall p \in P, \ \ M(p) \geq Pre(p,t). \tag{2.6}$$

That is, if the number of tokens in each input place is greater than or equal to the weight of the arc connecting this place to the transition.

The equation above can be written in the form

$$M \geq Pre(\,.\,,t) \tag{2.7}$$

where the column vector $Pre(.,t)$ represents the column of matrix Pre corresponding to transition t and M represents the initial marking vector.

There are other notations that express that t is enabled for a given marking M:

$$M(t> \quad M \xrightarrow{t} .$$

Example 2.2 Enabled transition
In the Petri net in FIG. 2.1, Example 2.1 for the initial marking:

$$M = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$$

with

$$Pre(.,a) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad Pre(.,c) = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$$

transitions a and c are enabled, as $M > Pre(.,a)$ and $M = Pre(.,c)$. FIG. 2.3 shows the enabled transitions a and c highlighted in red. This figure was obtained using Tina toolbox (Appendix E).

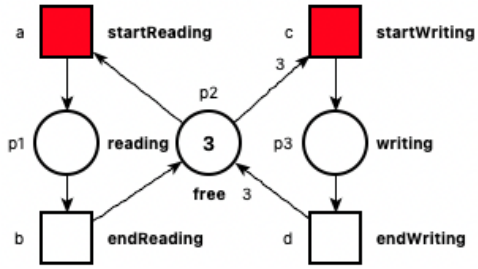


Figure 2.3: $M \xrightarrow{a}$ and $M \xrightarrow{c}$.

◇

2.1.6 Transition Firing

If t is enabled by a marking M, a new marking M' is obtained through the firing of t such that:

$$\forall p \in P, \quad M'(p) = M(p) - Pre(p,t) + Post(p,t). \tag{2.8}$$

The new marking M' is given by the equation

$$M' = M - Pre(.,t) + Post(.,t) = M + C(.,t). \tag{2.9}$$

The column vector $Pre(.,t_i)$ corresponding to the transition t_i in the matrix Pre can be written as $Pre[1]_i$, which is the product of the matrix Pre with the column vector $[1]_i$ (a vector with all elements zero except for the i-th element). Similarly, the column vector

$Post(., t_i)$ can also be written as $Post[1]_i$. The equation above can then be rewritten as:

$$M' = M - Pre[1]_i + Post[1]_i = M + C[1]_i. \tag{2.10}$$

The following notations are used to represent the marking obtained with the firing of t:

$$M(t > M'$$
$$M \xrightarrow{t} M'.$$

The firing of t is an operation that consists of removing $Pre(p, t)$ tokens from each preceding place p (weight of the input arc) and placing $Post(p, t)$ tokens in each following place p (weight of the output arc). By modifying the marking, the firing of t represents, in the model, the state change occurring in the system due to the occurrence of the event associated with transition t.

From equation 2.10, we can see that the matrix C updates the tokens when transition t is fired.

Each element $C(p_j, t_i) < 0$ of the column vector $C[1]_i$ indicates how many tokens are removed from each place p_j in the net when t_i is fired. The element $C(p_j, t_i) > 0$ indicates the number of tokens added to the marking of each place p_j when t_i is fired. If $C(p_j, t_i) = 0$, the marking of p_j is not changed.

Example 2.3 Transition firing
In the Petri net in FIG.2.3, Example 2.2, after firing transition a from the initial marking M, the next marking M_1 is obtained using equation 2.10:

$$\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}.$$

If, from the initial marking M, transition c were fired, the marking $M_2 = [0\ 0\ 1]^T$ would be obtained. Markings M_1 and M_2 are shown in FIG. 2.4 and 2.5, respectively. For each marking, new transitions are enabled, highlighted in red in the figure.

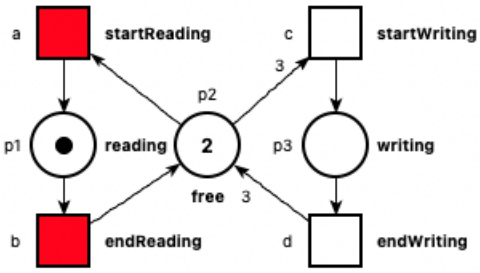


Figure 2.4: $M \xrightarrow{a} M_1$.

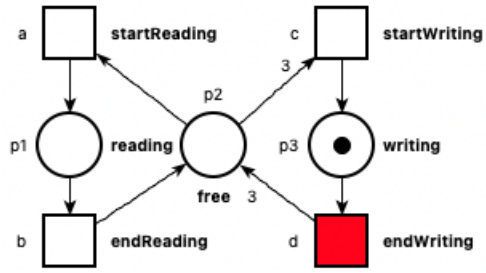


Figure 2.5: $M \xrightarrow{c} M_2$.

◇

2.1.7 Conflict and Parallelism

The notions of conflict and parallelism of transitions, presented informally in Section 1.5.2, will be defined next. Clearly understanding how each notion is represented in a Petri

net simplifies the task of system modeling. Once the interaction between activities is identified, it can be easily translated into the model.

Conflict can exist in the structure of the model without necessarily being realized; the marking must effectively allow its occurrence. The same applies to parallelism. Conflict occurs when there are, in a given state, two (or more) mutually exclusive possibilities for evolution. On the other hand, parallelism implies that all activities can be executed simultaneously.

Structural conflict: Two transitions t_1 and t_2 are in structural conflict if and only if they have at least one common input place:

$$\exists p \in P, \quad Pre(p, t_1)Pre(p, t_2) \neq 0. \tag{2.11}$$

Effective conflict: Two transitions t_1 and t_2 are in effective conflict for a marking M if and only if they are both structurally in conflict and are enabled:

$$M \geq Pre(\,.\,, t_1) \text{ and } M \geq Pre(\,.\,, t_2). \tag{2.12}$$

Structural parallelism: Two transitions t_1 and t_2 are structurally parallel if they do not have any common input place:

$$\forall p \in P \;\; Pre(p, t_1)Pre(p, t_2) = 0 \tag{2.13}$$

which is equivalent to checking if $Pre(\,.\,, t_1)^T \times Pre(\,.\,, t_2) = 0$, where $\times$ denotes the dot product of two vectors.

Effective parallelism: Two transitions t_1 and t_2 are in parallel for a given marking M if and only if they are structurally parallel and:

$$M \geq Pre(\,.\,, t_1) \text{ and } M \geq Pre(\,.\,, t_2). \tag{2.14}$$

The Petri net model allows for representing true parallelism. However, during implementation or simulation, this can become pseudo-parallelism. The key point is that the model supports this kind of representation. Regardless, parallel transitions can be fired independently of each other; the firing of one does not prevent the firing of the other(s), as is the case with conflicting transitions.

Example 2.4 Conflit and parallelism 1
In the Petri net of FIG. 2.1, transitions a and c are in structural conflict, as $Pre(p_2, a)Pre(p_2, c$ 3. For the initial marking M, transitions a and c are in effective conflict, because they are structurally in conflict and $M_1 > Pre(., a)$ and $M_1 > Pre(., c)$.

On the other hand, transitions b and d are structurally parallel. Indeed, as:

$$Pre(\,.\,, b) \;=\; \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad Pre(\,.\,, d) \;=\; \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

we have P $Pre(., b)^T \times Pre(., d) = 0$.

In the same Petri net structure as in FIG. 2.1, but now with the marking

$$M' = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

transitions b and d are effectively parallel for M'. Note that there is no more mutual exclusion between p_1 and p_3 in this model with such a marking. ◇

Example 2.5 Conflit and parallelism 2
In the Petri net depicted in FIG. 1.17.b, Example 1.2, there is a structural conflict between transitions t_e and t_g, which becomes effective when places p_4 and p_6 are marked. This indicates the presence of a batch of Pr_1 in buffer B_1 and a batch of Pr_2 awaiting processing by reactor R_2. This conflict models the shared use of resource R_2 (place p_9).

In the same Petri net, there is structural parallelism between transitions t_b and t_g, which is effective when places p_2 and p_6 are marked. This parallelism models the independent production processes of reactors R_1 (place p_3) and R_2 (place p_7). However, the structural parallelism between transitions t_b and t_e is not an effective parallelism. ◇

Example 2.6 Conflit vs. parallelism
Transitions t_2 and t_3 shown in FIG. 1.11 are structurally parallel and function effectively in parallel when marking $M(p_1) = 1$. Both transitions, highlighted in red in FIG. 2.6, are *enabled* by marking $M_1 = p_2p_3$. After the firing of t_2, leading to $M_2 = p_3p_4$ as shown in FIG. 2.7, t_3, highlighted in red, remains enabled. This is not the case for transition c in FIG. 2.4 in Example 2.3, which is in conflict with a, as c becomes disabled after the firing of a. ◇

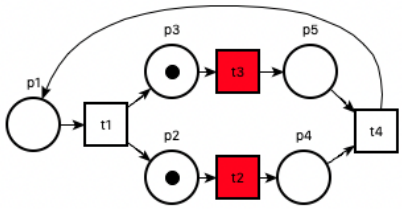


Figure 2.6: $M_1 \xrightarrow{t_2}$ and $M_1 \xrightarrow{t_3}$.

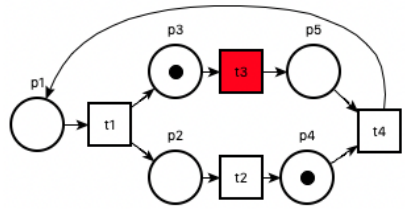


Figure 2.7: $M_1 \xrightarrow{t_2} M_2$ and $M_2 \xrightarrow{t_3}$.

2.1.8 Firing sequence

Note the Petri net in FIG. 2.8. Transition t_1 is enabled by the marking $M_0 = [1\ 0\ 0\ 0]$ and can fire, leading to the marking $M_1 = [0\ 1\ 0\ 0]$, $M_0 \xrightarrow{t_1} M_1$. After its firing, transition t_2 in turn can fire, leading to the marking $M_2 = [0\ 0\ 1\ 0]$, $M_1 \xrightarrow{t_2} M_2$. The sequential firing of transitions t_1 and t_2 is called a *firing sequence* and is denoted, in this case, by $s = t_1\,t_2$. It is said that the sequence $s = t_1\,t_2$ is fireable from M_0, with the following notation:

$$M_0 \xrightarrow{t_1\,t_2} M_2$$

or alternatively:

$$M_0(t_1\ t_2 > M_2.$$

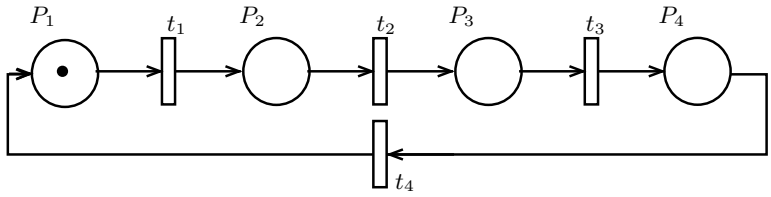


Figure 2.8: A sequence of transition firings.

Example 2.7 Firing sequence
Consider the Petri net in FIG. 2.1 in Example 2.2. The firing sequence $s = aab$, starting from marking M_0, leads the net to marking M_1 (FIG. 2.4), $M_0 \xrightarrow{aab} M_1$, successively passing through the markings:

$$\underset{M_0}{\begin{bmatrix}0\\3\\0\end{bmatrix}} \xrightarrow{a} \underset{M_1}{\begin{bmatrix}1\\2\\0\end{bmatrix}} \xrightarrow{a} \underset{M_2}{\begin{bmatrix}2\\1\\0\end{bmatrix}} \xrightarrow{b} \underset{M_1}{\begin{bmatrix}1\\2\\0\end{bmatrix}}.$$

◇

2.1.8.1 Characteristic Vector

Consider the vector $\mathbf{s}$ where each component $\mathbf{s}(t)$ represents the number of occurrences of transition t in a firing sequence s. This vector is called the *characteristic vector* of the sequence s. Its dimension is equal to the number of transitions in the Petri net.

The evolution of the marking of a Petri net, with initial marking M, for a sequence $s = t_i t_j \dots t_l$, applying successively the equation 2.10, is given by:

$$\begin{aligned} M' &= (\dots((M + C[1]_i) + C[1]_j) + \dots + C[1]_l \\ &= M + C([1]_i + [1]_j + \dots + [1]_l). \end{aligned} \tag{2.15}$$

The sum of the vectors $[1]_i + [1]_j + \dots + [1]_l$ corresponds exactly to the characteristic vector $\mathbf{s}$ of the sequence s, indicating how many times each transition of the net has been fired. Thus:

$$M' = M + C\mathbf{s} \quad \text{com} \quad M \geq 0,\ \mathbf{s} \geq 0. \tag{2.16}$$

This equation is called the *fundamental equation* of the Petri net.

Example 2.8 Characteristic Vector
The characteristic vector $\mathbf{s}$ of the sequence $s = aab$ in the Example 2.7 is given by:

$$\mathbf{s} = \begin{bmatrix}2\\1\\0\\0\end{bmatrix} \begin{matrix}a\\b\\c\\d\end{matrix}.$$

$s(a) = 2$ indicates that transition a was fired twice in the sequence s, and $s(b) = 1$ means that transition b was fired only once. The null values of $s(c)$ and $s(d)$ indicate that transitions c and d were not fired in this sequence $s = aab$. ◇

2.1.8.2 Considerations on the existence of s

In Petri net theory, characterizing the marking with a vector M is precise as it provides all the necessary information about the state. However, the representation of a sequence s through the vector $\mathbf{s}$ disregards the order of transition firing. Indeed, the vector s results from the sum $[1]_i + [1]_j + \ldots + [1]_l$; since addition is a commutative operation, any permutation in the firing order of transitions t_i, t_j $\ldots$ t_l within the sequence will yield the same vector $\mathbf{s}$. Consequently, the firing order is lost, resulting in an information loss regarding the evolution of the net. Only the transformation of the marking is described.

Therefore, it should come as no surprise that the existence of a characteristic vector $\mathbf{s}$, a solution of Equation 2.16, does not guarantee that the sequence s can actually be fired, and thus that M' exists. In fact, it is necessary for the initial marking to be such that the transitions are indeed fired for each intermediate marking. The designer needs to check all possible combinations of the characteristic vector $\mathbf{s}$ with a Petri net simulator or by successively applying Equation 2.9. It can be tedious, but at least it provides the (reduced) set of transitions that *could* be fired.

Example 2.9 Firing order of s
Consider the net in Figure 2.8: for the marking M_0 in the figure, the sequence $t_1\,t_2$, represented by the vector $\mathbf{s}^T = [\,1\;1\;0\,]$, is fireable (leading to marking M_2), whereas the sequence $t_2\,t_1$, represented by the same vector $\mathbf{s}$, cannot be fired from M_0. ◇

Example 2.10 s and non-fireable sequence
Consider the Petri net in FIG. 2.9. The matrix C is shown in FIG. 2.10.

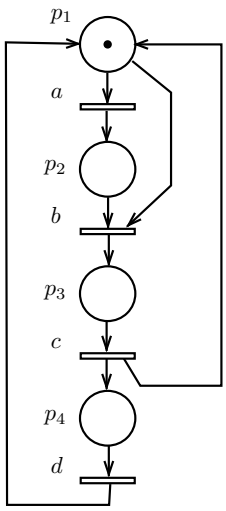


Figure 2.9: Petri net with non-firable sequence.

$$C = \begin{bmatrix} -1 & -1 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Figure 2.10: C matrix.

We want to know if the sequence $s = abcd$, represented by the vector $\mathbf{s}$, is fireable from the initial marking M,

$$M = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{s} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Applying equation 2.16, we get the marking

$$M' = M + C\mathbf{s} = [1\ 0\ 0\ 0]^T.$$

Although the value of M' is positive, this marking cannot be reached by firing the sequence $s = abcd$ from M. Just use equation 2.8 to verify that, from M, after firing a, no other transition can fire. Thus, the existence of a marking $M' > 0$ through equation 2.16 does not imply the existence of the sequence s that would actually generate it. If $M = [2\ 0\ 0\ 0]^T$ in the Petri net of FIG. 2.9, we obtain, applying equation 2.16 with the same vector $\mathbf{s}$ above, the marking $M' = M$. In this case, the sequence $s = abcd$ (in this order) is indeed fireable, $M \xrightarrow{abcd} M'$. However, any other sequence obtained by permuting the order of the transitions is not. ◇

The questions that arise are as follows:

- Given an initial marking M and a sequence s, is there a marking M' such that $M \xrightarrow{s} M'$? If $M' < 0$ (equation 2.16) there is no such sequence. If $M' > 0$, there is also no guarantee of the existence of s. At best, the order of firing the transitions is unknown. In the example of FIG. 1.9, for $M_0 = [1\ 0\ 0]^T$, the sequence $s = t_1t_2t_3$ is fireable, but $s = t_2t_3t_1$ is not. And both are characterized by the same vector $\mathbf{s}^T = [\ 1\ 1\ 1\ ]$;

- Given an initial marking M and any arbitrary marking M', is there an s such that $M \xrightarrow{s} M'$? If $s < 0$ there is no such sequence. But if $s > 0$ there is no guarantee of the existence of s. The particular case where $M' = M$ in which s is such that $C\mathbf{s} = 0$ leads to the definition of a transition invariant in Section 2.5.

- Given an initial marking M and a known marking M', does the existence of a vector $s > 0$ imply the existence of all sequences s obtained by permuting the firing order of transitions) representable by this vector? Since the order is not known, some sequences — for a given marking — may exist and others may not.

The characterization of firing sequences in Petri net theory is not precise. Chapter 8 demonstrates how to obtain a precise characterization of a sequence using linear logic.

2.1.9 Set of Reachable Markings

The set of reachable markings $A(R, M)$ of a marked Petri net is the collection of markings that can be reached from the initial marking through a sequence of firings:

$$A(R, M) = \{M_i,\ \exists s\ \ M \xrightarrow{s} M_i\}. \tag{2.17}$$

If this set is finite, it can be represented as a graph $GA(R, M)$, described by Algorithm 1, where the nodes are the reachable markings of $A(R, M)$. A directed edge connects two nodes M_i and M_j if there exists a transition t that, when enabled, allows the transition from marking M_i to M_j: $M_i \xrightarrow{t} M_j$. This edge is labeled with the transition t.

The graph of reachable markings is the state machine *equivalent* to the Petri net. However, it's important to note that, since the notion of process disappears, it is no

longer possible to differentiate between parallel transitions and conflicting transitions for a given marking within the graph. This can be observed in Example 2.12.

Algorithm 1 Marking Graph $GA(R, M)$

Read the Petri net input file $\{R =< P, T, Pre, Post >$ and initial marking $M_0\}$
Reachable := $\{M_0\}$
ToExplore := $\{M_0\}$
Graph := ϕ
while ToExplore $\neq \phi$ **do**
  M := a marking from set ToExplore
  ToExplore := ToExplore - {M}
  for all transitions $t \in T$ **do**
    if M > Pre(.t) **then**
      M' := M + Post (.,t) - Pre (.,t)
      if M' $\notin$ Reachable **then**
        Reachable := Reachable $\cup \{M'\}$
        ToExplore := ToExplore $\cup \{M'\}$
      end if
      Graph := Graph $\cup (M, M', t)$
    end if
  end for
end while

Example 2.11 Reachable markings 1
The FIG. 2.11 depicts the graph of reachable markings for the Petri net shown in FIG. 2.1, Example 2.1, with the initial marking $M = [0\ 3\ 0]^T$.

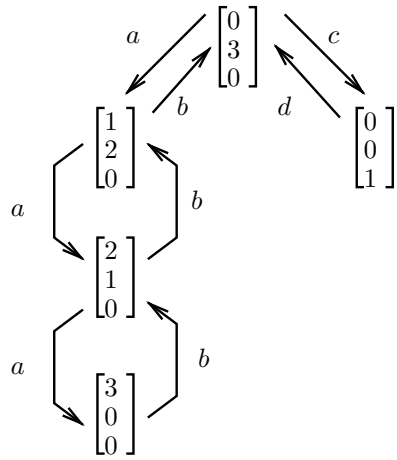


Figure 2.11: Graph of reachable markings.

◇

Example 2.12 Reachable markings 2
The marking graph of the Petri net shown in FIG. 1.17 in Example 1.2 has the same graph as the global state machine represented in FIG. 1.18 where:
$M_0 = p_1p_6p_8p_9 = S_0 : Pr_1$ and Pr_2 waiting, reactors free
$M_1 = p_2p_6p_8p_9 = S_1 : Pr_1$ in B_1, Pr_2 waiting, reactors free
$M_2 = p_4p_6p_8p_9 = S_2 : Pr_1$ in B_2, Pr_2 waiting, reactors free
$M_3 = p_1p_7p_8 = S_3 : Pr_1$ waiting, R_2 processing Pr_2, R_1 free
$M_4 = p_3p_6p_9 = S_4 : R_1$ processing Pr_1, Pr_2 waiting, R_2 free
$M_5 = p_2p_7p_8 = S_5 : Pr_1$ in B_1, R_2 processing Pr_2, R_1 free
$M_6 = p5p_6p_8 = S_6 : R_2$ processing Pr_1, Pr_2 waiting, R_1 free
$M_7 = p_4p_7p_8 = S_7 : Pr_1$ in B_2, R_2 processing Pr_2, R_1 free
$M_8 = p_3p_7 = S_8 : R_1$ processing Pr_1, R_2 processing Pr_2.

2.2 Petri Net Simulator

Algorithm 2 presents the Petri net token player, which allows for simulating an ordinary (or classical) Petri net. It is a general program that updates the marking each time a transition t is fired. Starting from an initial marking, the token player verifies if there are *enabled* transitions. When there is no conflict, the enabled transition is *fired*. When there is a conflict, a transition must be chosen to resolve the non-determinism. For example, a priority could be assigned to transitions, and the transition in the set $\mathcal{E}$ of enabled transitions with the highest priority will be fired using Equation 2.9. The firing of a transition can disable some transitions that were enabled at the initial marking, so after a transition firing it is necessary to verify again the new set of enabled transitions. The non-determinism can be resolved by associating supplementary firing conditions, as it will be seen in Chapter 4.

Algorithm 2 Petri Net Token Player

```
Read the input file {PN R =< P, T, Pre, Post > and M0}
deadlock =False
M = M0 {Start from initial marking}
while not(deadlock) do
   Calculate the set E of enabled transitions of the PN (Eq. 2.6)
   if E ≠ φ then
      if | E |> 1 then
         Choose a transition t ∈ E
      end if
      Fire transition t (Eq. 2.9) {Update marking M}
   else
      deadlock = True
   end if
end while
```

2.3 Petri Net and Rule-Based Systems

The Petri net can be represented either graphically or in matrix form, as presented in the preceding section. Given its characteristic of evolving the state based on rules

represented by transitions, the Petri net can also be considered as a system of production rules based on a representation in the form:

if condition *then* action.

2.3.1 Rule-Based system

A knowledge production rule system, or simply a rule-based system, consists of:

- a fact base, representing the available knowledge about the system;
- a rule base, allowing the deduction of new facts;
- an inference mechanism, called the *inference engine*, which allows making new deductions by applying the rules to the facts.

The most elementary inference mechanism consists of arranging the set of rules in a list and sequentially traversing it. If the *condition* part of the rule is true in the current context (corresponds to a fact in the fact base), the rule is applied or fired. If no rule is applicable or if a terminal fact (desired conclusion) is deduced, the inference mechanism stops.

Most of the time, in a given context, several rules can be applied. The result of deduction, naturally, can be different depending on the chosen rule. The most elementary inference mechanism selects the first rule without checking if there are others that could also be applied.

A situation where multiple rules can be applied constitutes a *conflict*. Resolving a conflict is carried out by the *control* of the inference engine, which can be quite complex.

The correspondence between Petri nets and rule-based systems is based on the following:

- The set of transitions in the Petri net, with their conditions and actions represented, respectively, by the vectors $Pre(.,t)$ e $Post(.,t)$ (Section 2.1.5), constitutes the rule base.
- The initial marking corresponds to the initial fact base or initial context.
- Control is given by the structure of the Petri net: if transitions are parallel, the firing order is indifferent; if transitions are in effective conflict, only one can be fired.

A rule-based system, unlike a system described using classical logic, does not allow for formal verification. However, it allows for representing knowledge-based systems and obtaining results when no known algorithm exists.

Rule-based systems can be monotone or non-monotone AKERKAR (2009).

- Monotone (deduction): what is true remains true forever; you only add new facts, so there is no possible conflict.
- Non-monotone (state changes); possible conflicts and non determinism arise if several rules apply, leading to different state changes.

A Petri net can be viewed as a non-monotone rule-based system, where the token player corresponds to the inference engine. The analysis, proof, and verification is an additional mechanism that can enhance a rule-based sysem, e.g. for model checking TSAI (1999). Chapter 3 will present analysis, validation and verification for Petri nets. The verification using model checking is not covered in this book.

2.3.2 Grammar

A grammar is a system for rewriting words from a given alphabet. Let $S =< I\!P, Q >$ be the grammar associated with the Petri net $R =< P, T, Pre, Post >$, defined by:

- An alphabet $I\!P$ whose symbols are the places $p \in P$, and
- A set Q of rewriting rules

$$t : \mu(Pre(., t)) \rightarrow \mu(Post(., t)). \tag{2.18}$$

Let $I\!M$ be the set of all markings of the Petri net. The application $\mu : I\!M \rightarrow I\!P^*$ associates, with a marking vector of the net, a word from the vocabulary $I\!P^*$. $I\!P^*$ is the set of finite sequences of elements from P, including the empty sequence λ. For each marking $M(p) = i$, $i \in I\!N$, of the places $p \in P$ of the net, an occurrence of a symbol from $I\!P^*$ is associated; the word $\mu(M)$is obtained by concatenating the symbols p^i. For the sake of simplicity, this word is also represented by M in this book.

In general, an application $\mu : I\!N^m \rightarrow I\!P^*$ associates, with a vector v from $I\!N^m$, a word $\mu(v)$ from $I\!P^*$.

A transition t is enabled by a marking M if and only if:

$$\mu(Pre(., t)) \subseteq \mu(M). \tag{2.19}$$

The firing of t leads to a new marking M', given by:

$$\mu(M') = \mu(M) \cup \mu(Post(., t)) \setminus \mu(Pre(., t)). \tag{2.20}$$

The expression $v \cup w$ indicates the concatenation of words v and w, and $v \setminus w$ denotes the exclusion of the word w from the word v.

An observation is warranted here: the order of letters in words from the vocabulary $I\!P^*$ is not important. For the sake of readability, places are typically written in alphanumeric order.

A firing sequence s can also be represented as a word in the vocabulary $I\!T^*$, formed from an alphabet of transitions $t \in I\!T$.

Unlike the words in the vocabulary $I\!P^*$, the order of the letters $t \in T$ in a word belonging to $I\!T^*$ is important, as it indicates the firing order of t in the sequence s.

Example 2.13 Grammar
Consider the Petri net depicted in FIG. 2.1, Example 2.1. The corresponding grammar is defined by:

- The alphabet $I\!P = \{p_1, p_2, p_3\}$, and

- The set of rewriting rules: $Q = \begin{cases} a : p_2 \rightarrow p_1 \\ b : p_1 \rightarrow p_2 \\ c : p_2^3 \rightarrow p_3 \\ d : p_3 \rightarrow p_2^3 \end{cases}$.

The initial marking $M = [\,0\ 3\ 0\,]^T$ of the net corresponds to the axiom $\mu(M) = p_2^3$ of the grammar, from which new words can be derived. For sake of simplicity, we will use the notation $M = p_2^3$.

Equation 2.19 allows us to verify that, for the marking $M = p_2^3$, only transitions a and c are enabled, as $\mu(Pre(.,a)) = p_2 \subseteq M$ and $\mu(Pre(.,c)) = p_2^3 \subseteq M$. If transition a is fired, we obtain, by applying Equation 2.20:

$$M' = p_2^3 \cup p_2 \setminus p_1 = p_2^2\, p_1 = p_1\, p_2^2.$$

Similarly, if transition c is fired from M, we obtain:

$$M'' = p_2^3 \cup p_3 \setminus p_2^3 = p_3.$$

By applying equation 2.20 successively, it is possible to derive all the words corresponding to the reachable markings from the initial marking. Thus, the grammar provides a formalism for generating the set of reachable markings of a Petri net:

$$A(R, M) = \{p_2^3, p_1 p_2^2, p_1^2 p_2, p_1^3, p_3\}.$$

You can compare this set with the marking graph presented in Example 2.11.

Starting from the initial marking $M = [\,0\ 3\ 0\,]^T$, or axiom $M = p_2^3$, the sequences a^3, a^2b, $abab$, and cd (among others) are part of the effective vocabulary of the net, while the sequences ba, dc, ac, and ca (among others) are prohibited. ◇

Although the rewriting rule $a : p_1 \rightarrow p_2 p_3$ is quite similar to the propositional logic formula $p_1 \rightarrow p_2 \wedge p_3$, the Petri net cannot be used as a semantics for (monotone) classical logic. This topic is addressed in chapter 8.

2.4 Properties of the Model

In this section, we will define the properties related to the marked Petri net: liveness, boundedness, and reversibility, grouped together under the generic term of "good properties." Their definitions involve considerations about the set of reachable markings from the initial marking. Marked Petri nets do not directly provide algorithms capable of determining whether a property is verified, as the set of reachable markings is not always finite. The methods for analyzing these properties based on the marking graph will be presented in Chapter 3.

The properties related to unmarked Petri nets (which therefore do not depend on their initial marking) allow deriving calculation methods directly from the definitions through the resolution of a system of linear equations. These properties will be defined in Section 2.5 concerning conservative and repetitive components.

2.4.1 k-Bounded Marked Net

2.4.1.1 k-Bounded Place and Binary Place

A place p in a marked net N is k-bounded if and only if:

$$\forall M' \in A(R, M), \quad M'(p) \leq k. \tag{2.21}$$

If $k = 1$ the place is said to be *binary* or *safe*.

Example 2.14 Binary place
In the Petri net shown in FIG.2.1, whose marking graph is represented in FIG.2.11, for the initial marking $M = {p_2}^3$ (or $M = [0\ 3\ 0]^T$), place p_3 is binary, while places p_1 and p_2 are 3-bounded. ◇

2.4.1.2 k-Bounded Marked Petri Net and Binary Marked Net

A marked net N is k-bounded if and only if all its places are k-bounded.

A marked net N is *binary* if and only if all its places are binary. It is also referred to as *safe*.

The Petri net in FIG. 2.1, for the marking $M = {p_2}^3$, is 3-bounded, or bounded with $k = 3$. If its initial marking were $M = p_2$, the net would be binary (however, transitions c and d would never fire).

Example 2.15 Unbouded place
Consider the Petri net in FIG. 2.12, with the initial marking $M = p_1$. Each time the sequence $s = ab$ is fired, a token is introduced into place p_3. Therefore, this place is unbounded, and consequently, the net is also unbounded. The places p_1 and p_2 are also unbounded.

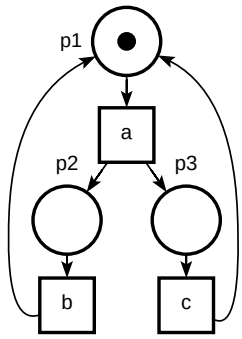


Figure 2.12: Unbounded Petri net.

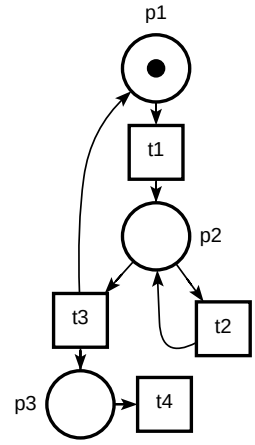


Figure 2.13: Unbounded Petri net (p_3).

The Petri net in FIG. 2.13 is unbounded because of place p_3, but places p_1 and p_2 are bounded.

An example of an unbounded net that describes a classic mechanism is shown in FIG. 2.14. If the opening of a parenthesis (left parenthesis) is associated with the firing of transition a and the closing of a parenthesis (right parenthesis) with the firing of

transition b, the sequences of transition firings that lead from the initial marking $M = p_1$, to the same marking M, describe all allowed parenthesized expressions.

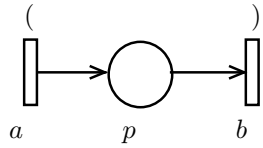


Figure 2.14: Parenthesis Petri net.

Clearly, this net is unbounded because it is always possible to open another parenthesis (fire a) and thus place another token in place p_1. ◇

Regarding the property of a net being bounded, or specifically, safe, it is important to note that:

- The Petri net considered here always has a given initial marking (marked net).
- The concept of a bounded net corresponds to the fact that a physical system is always bounded. However, one might use an unbounded Petri net when evaluating the performance of a system independently of the limits of its intermediate storage elements.
- The concept of a safe net has the following significance: if places represent logical conditions, the presence of more than one token in a place indicates that an inconsistency has been introduced into the model. Generally, this is a logical condition set to *1* during one cycle of operation that was not used (was not set to *0*) and is set to *1* again in the next cycle.
- In a safe net, only one process is represented. Any (effectively) parallel transitions that occur are the result of temporary divisions in the process, which later synchronize (see the discussion in section 2.1.7).

2.4.2 Live Marked Net

2.4.2.1 Quasi-Live Transition

A transition t of a marked net $N =< R, M >$ is quasi-live if and only if there exists a firing sequence s such that

$$\exists s \mid M \xrightarrow{s} M' \quad \text{and} \quad M' \xrightarrow{t} . \tag{2.22}$$

In other words, transition t is quasi live if it can be enabled by a marking M' in the graph of reachable markings, obtained from the initial marking M through the firing of a sequence of transitions s, possibly empty ($s = \lambda$). It is said that t can be enabled from M through the firing of the sequence s. The above equation can be rewritten in the form $\exists s \mid M \xrightarrow{st}$.

2.4.2.2 Live Transition

A transition t of a marked net $N =< R, M >$ is live if and only if:

$$\forall M' \in A(R, M),\ \exists s \mid\ M' \xrightarrow{st} . \tag{2.23}$$

To be live, the transition t must be enabled from any marking M' in the graph of reachable markings through a firing sequence s.

Example 2.16 Quasi-live
Consider the Petri net in FIG. 2.15. The graph of reachable markings associated with the initial marking $M = p_3\, p_4$ is given by FIG. 2.16. It is easily seen that transition d is quasi-live, as it can be fired once (for the marking $M = p_3\, p_4$). However, it is not live: after this firing, d is blocked and no longer enabled by any marking. On the other hand, transitions a, b, c, e, and f are live, because for any marking accessible from M, it is possible to find a firing sequence s that leads to a marking that enables them.

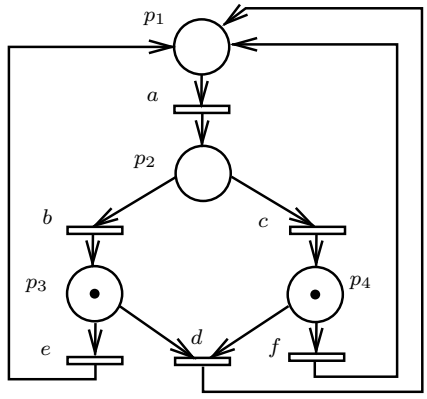


Figure 2.15: Quasi-live and non-live transition.

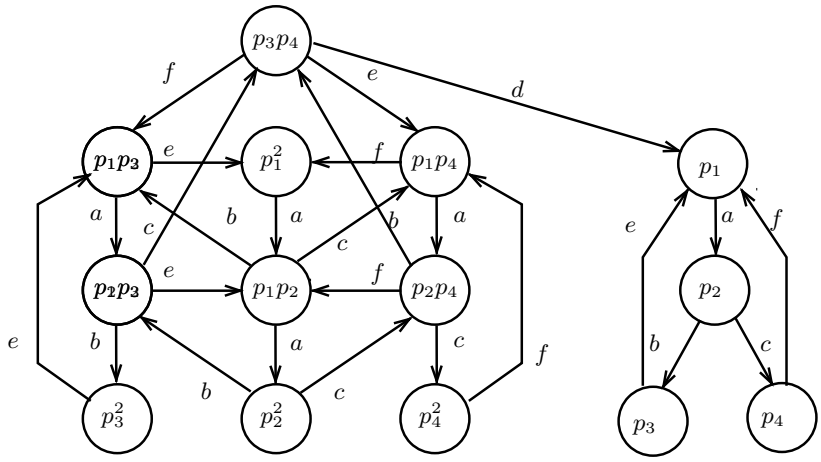


Figure 2.16: Marking graph (quasi-live transition).

◇

Example 2.17 Quasi-live transition in an infinite sequence
One must be careful because a transition may, without being live, appear an infinite number of times in infinite sequences of transition firings. This is the case for transition g in FIG. 2.17. The marking graph is the same as in FIG. 2.16 with a new arc labeled by transition g from marking p_3p_4 to marking p_3p_4. It can always be fired before the firing of d, but it cannot be fired anymore after the firing of d.

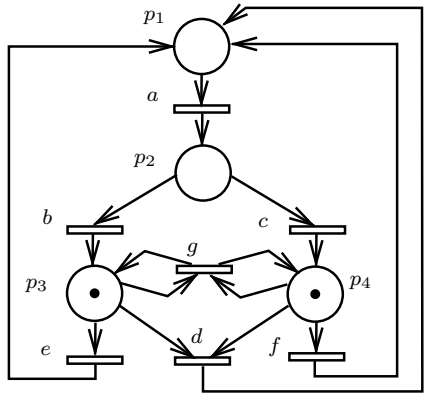


Figure 2.17: Quasi-live transition and infinite sequence.

◇

2.4.2.3 Live marked net

A marked Petri net $N =< R, M >$ is live if and only if all its transitions are live:

$$\forall t \in T,\ \forall M' \in A(R, M),\ \exists s \mid\ M' \xrightarrow{st} . \tag{2.24}$$

It is important to note that:

- Only the marked Petri net is considered here.
- A live Petri net ensures that no deadlock can be caused by the structure of the net. However, it does not prove the absence of potential deadlocks caused by a poor interaction between the Petri net and its external environment (see later in Chapter 4 the problems related to the interpretation of the Petri net).
- A live Petri net also guarantees the absence of dead parts (never reached), with the same observation regarding interpretation.

Example 2.18 Liveness
The Petri net in FIG. 2.1 from Example 2.1 is live for the initial marking $M = [\,0\,3\,0\,]^T$ (or $M = p_2^3$). However, the same Petri net with the marking $M = [\,0\,1\,0\,]^T$ (or $M = p_2$) is not live because c and d are never fired.

The net in FIG. 2.15, discussed in Example 2.16, for $M = p_3p_4$, is not live, as transition d is not live (it is quasi-live).

A Petri net can be unbounded and live: for example, the Petri net in FIG. 2.14 from Example 2.15, where the sequence $s = a\,b$ is always enabled regardless of the marking. ◇

2.4.3 Reversible marked net

A marked net $N =< R, M >$ is reversible if and only if:

$$\forall M' \in A(R, M), \quad \exists s \mid \quad M' \xrightarrow{s} M. \tag{2.25}$$

Thus, it is possible, starting from any reachable marking M' of $GA(R, M)$, to find a firing sequence s that brings the net back to the initial marking M. Most systems have repetitive operations and, therefore, the Petri nets used to represent them are reversible.

Example 2.19 Reversibility
Consider the marked Petri net in FIG. 2.1, whose reachable markings are shown in FIG. 2.11 This net is bounded, live, and reversible, as is the Petri net in FIG. 2.18. The Petri net in FIG. 2.19 is bounded and reversible, but it is not live. A bounded, non-reversible and non-live Petri net is depicted in FIG. 2.20.

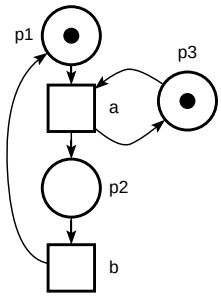


Figure 2.18: Bounded, reversible and live.

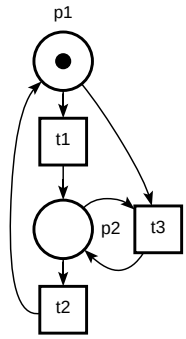


Figure 2.19: Bounded, reversible, but not live.

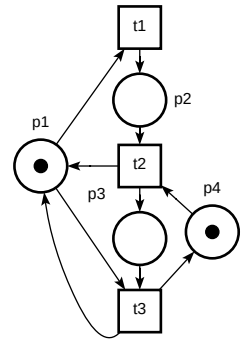


Figure 2.20: Bounded, not reversible, not live.

◇

Relaxed reversibility: home state M'

A marked net $N =< R, M >$ have a home state M' if:

$$\exists s' \mid \ M_0 \xrightarrow{s'} M' \mid \forall M \in A(R, M'), \ \exists s \mid \ M \xrightarrow{s} M' \tag{2.26}$$

It is possible to get back to the home state M' from whatever marking M in the marking graph obtained from M', provided that it is possible to reach M' from the initial marking M_0.

Example 2.20 Home state
Consider the marked Petri net in FIG. 2.21.a, whose reachable markings are shown in Figure 2.21.b. This net is non-reversible, as there is no sequence of firings that allows returning to the initial marking $M = p_1p_4$ after firing transition a. However, this net has a home state, $M' = p_2p_4$. It is interesting to note that the Petri net is live, as from any reachable marking from $M_0 = p_1p_4$, there exists a firing sequence leading to a marking that enables each transition of the net.

◇

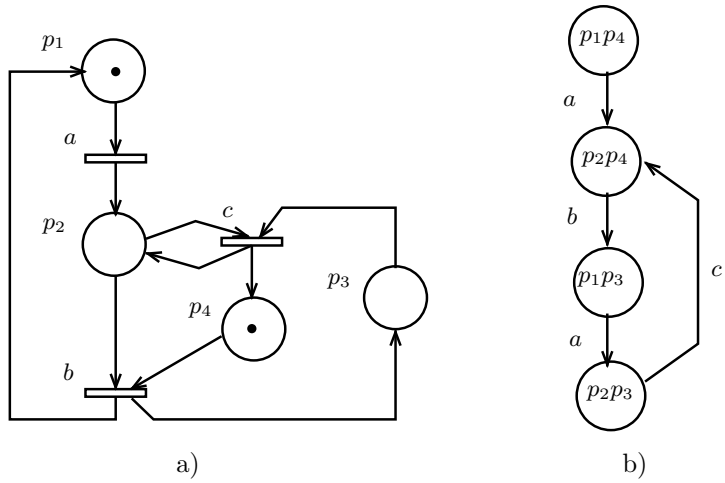


Figure 2.21: a) Non-reversible Petri net; b) Associated marking graph.

Once again, it is important to emphasize that the properties that have been defined are strongly related to the initial marking as shown in Example 2.21.

Example 2.21 The role of the initial marking
The Petri net in Example 2.20 is not reversible for the marking $M_0 = p_1p_4$, but for any of the following initial markings – $M = p_2p_4$, $M = p_1p_3$, or $M = p_2p_3$ – the net is, both live and reversible.

Conversely, the Petri net in Example 2.16 is not reversible for the initial marking $M = p_3p_4$, but it is reversible for the marking $M = p_1$. However, it is not live, as transition d will never be fired.

Consider the Petri net in Figure 2.22. For the marking $M = p_1p_3$, the net is binary, live, and reversible. However, if a token is added to place p_4, the net ceases to be bounded (see the sequence $abab...$). Finally, for the marking $M = p_1$, the net is binary but not live (no transition is enabled).

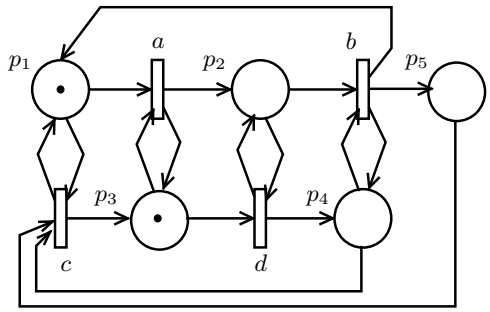


Figure 2.22: Net with properties depending on the initial marking.

◇

2.5 Structural Properties

In this section, properties derived directly from the *structure* of the Petri net will be considered, which, therefore, do not depend on its initial marking. These properties are defined through the conservative place components and repetitive components. From these structural elements, one can also use information about the marking, thus defining the place and transition invariants, which provide some additional information about the dynamic behavior of the Petri net.

2.5.1 Conservative Components, Place Invariant

Motivation

Consider the Petri net in FIG. 2.23 described in Example 2.1. It is a modified version of the model in FIG. 2.1, where one reader is waiting in place p_1 and one writer is waiting in place p_5.

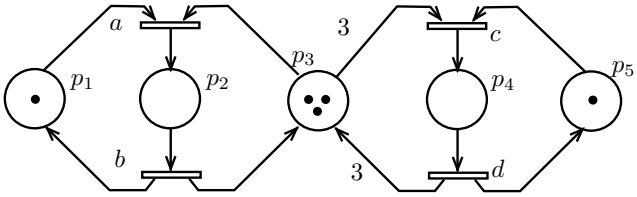


Figure 2.23: Readers and writers.

$$C = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

Figure 2.24: Matrix C of FIG 2.23.

Observe the circuit formed by places p_1 and p_2 and transitions a and b in FIG. 2.23. It is also represented in FIG 2.25. The sum:

$$M(p_1) + M(p_2)$$

equals 1 for the initial markingl $M_0 = [1\ 0\ 3\ 0\ 1]^T$. The firing of a does not change this sum, just like the firing of b, although the marking of each place is modified with each transition firing. The firing of the other transitions in the net (c and d) also does not modify this sum. For this example, it can be verified that, for all markings reachable from the initial marking, we have $M(p_1) + M(p_2) = 1$, or more generally:

$$\forall M \in A(R, Mo), \quad M(p_1) + M(p_2) = M_o(p_1) + M_o(p_2).$$

The linear form $M(p_1) + M(p_2) = M_o(p_1) + M_o(p_2)$ is called a *place invariant*, as the sum (generally weighted) of tokens is conserved for these places. The set of places p_1 and p_2 forms a *conservative component* of the net.

Considering now the circuit formed by places p_2, p_3 and p_4 and transitions a, b, c and d in FIG. 2.23, and also in FIG 2.25, one can verify that the set p_2, p_3 and p_4 forms a conservative component, with the place invariant:

$$\forall M \in A(R, M_0), \quad M(p_2) + M(p_3) + 3.M(p_4) = 3.$$

A *place invariant* is a linear function of the marking of places whose value is a constant that depends only on the initial marking of the net. It corresponds to a restriction on the states and activities of the system that will always be satisfied, regardless of its evolutions.

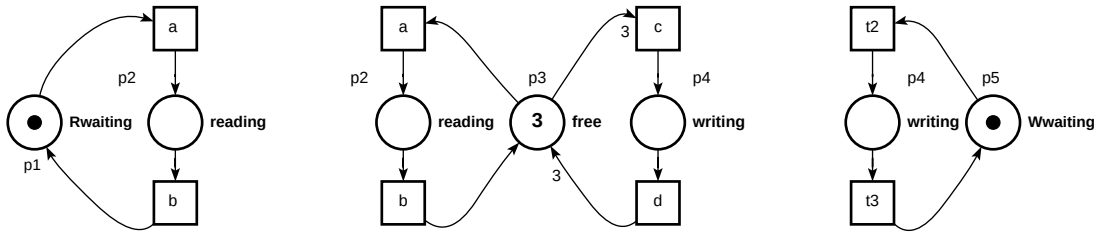


Figure 2.25: Circuits in FIG. 2.23.

Definition
The equation that allows determining the evolution of the net is the fundamental equation (Equation 2.16). To obtain a weighted sum of the markings, the equation is pre-multiplied by a vector $\mathbf{f}^T$:

$$\mathbf{f}^T M' = \mathbf{f}^T M + \mathbf{f}^T C \mathbf{s}. \tag{2.27}$$

It is easy to verify that the only way to make the equation independent of the firing sequences s is to nullify the product $\mathbf{f}^T C$.

A *conservative component* of a Petri net is the set of places $p_i \in P$ corresponding to the non-zero elements f_i of the column vector $\mathbf{f}_{n\times 1}$, the solution of the equation:

$$\mathbf{f}^T C = 0 \;\text{ with }\; \mathbf{f} > 0. \tag{2.28}$$

Notation:
For the sake of readability, the vector $\mathbf{f}^i$ of conservatives components can be written as a set of places $f^i = \{p_j \mid f^i_j \neq 0\}$. The set notation can also be omitted. See Example 2.22.

From a graphical point of view, a *conservative component* defines a sub-Petri net. Only places for which the corresponding component of $\mathbf{f}$ is non-zero, and the input and output transitions for these places, are considered. The transitions have the same number of input and output arcs (in simple cases).

A Petri net is said to be *conservative* if all places in the net belong to a conservative component.

If $\mathbf{f}$ is a solution of equation 2.28, then the linear function:

$$\mathbf{f}^T M = \mathbf{f}^T M_0 \;\; \forall M \in A(R, M_0), \tag{2.29}$$

is the corresponding *place invariant*, also called a *p-invariant*.

The place invariant of a Petri net, obtained through equation 2.29, depends on the initial marking, whereas the conservative component, obtained from equation 2.28, is completely independent, since the structure, represented by matrix C, is the same.

The integer solutions of Equation 2.28, $\mathbf{f}^T C = 0$ with $\mathbf{f} > 0$, are also called p-flow and can be calculated using the Gauss algorithm (see Appendix B). If all the coefficients of a flow are positive, it is called a p-semi-flow and can be calculated using the Farkas algorithm.

The place invariant allows, without enumerating all reachable markings, obtaining some information about the boundedness property of the Petri net, as will be seen in chapter 3.

Example 2.22 Conservative components
The conservative components of the Petri net in FIG. 2.23, obtained applying Equation 2.28 with the matrix C shown in FIG. 2.24, are:

$$\mathbf{f}^1 = [1\ 1\ 0\ 0\ 0]^T, \mathbf{f}^2 = [0\ 0\ 0\ 1\ 1]^T, \mathbf{f}^3 = [0\ 1\ 1\ 3\ 0]^T,$$

and can be written using the notation introduced in page 2.5.1 as: $f^1 = p_1\ p_2$, $f^2 = p_4\ p_5$, $f^3 = p_2\ p_3\ p_4^3$.

They define the sub-Petri nets in FIG 2.25. ◇

Example 2.23 Place invariants
The linear place invariants corresponding to the conservative components of Example 2.22 obtained applying Equation 2.29 for the values of $\mathbf{f}^1$, $\mathbf{f}^2$, and $\mathbf{f}^3$, are:

$$\begin{array}{ll} I_1: & M(p_1) + M(p_2) = M_0(p_1) + M_0(p_2) = 1 \\ I_2: & M(p_4) + M(p_5) = M_0(p_4) + M_0(p_5) = 1 \\ I_3: & M(p_2) + M(p_3) + 3 * M(p_4) = M_0(p_2) + M_0(p_3) + 3 * M_0(p_4) = 3. \end{array}$$

Note that these invariants provide information about the processes in the Petri net:

- I_1: describes the behavior of the reader: it is waiting or reading.
- I_2: describes the behavior of the writer: it is waiting or writing.
- I_3: describes the behavior of the resource: it is reading or writing.

Consider the Petri net in FIG. 2.23 again, but with another initial marking, $M_0' = [3\ 0\ 3\ 0\ 1]^T$, where three readers are waiting in place p_1. Invariants I_2 and I_3 do not change, but I_1 is now:

$$I_1 : M(p_1) + M(p_2) = M_0'(p_1) + M_0'(p_2) = 3.$$

There are three readers that are waiting (p_1) or reading (p_2). Different state are allowed: one reading and two waiting, two reading and one waiting, three reading, or three waiting. ◇

Example 2.24 Vector f and place invariants
Let us consider the example of the Petri net in FIG. 2.21.a. The matrix C is given by:

$$C = \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

Using equation 2.28, we find $\mathbf{f}^1 = [1\ 1\ 0\ 0]^T$ and $\mathbf{f}^2 = [0\ 0\ 1\ 1]^T$. Therefore, the net is conservative since $f(p) \neq 0$ for every place p.

Substituting the values of $\mathbf{f}^1$ and $\mathbf{f}^2$ into equation 2.29, for the initial marking $M_0 = [\,0\ 1\ 0\ 1\,]^T$, we find the place invariants:

$$I_1: \quad M(p_1) + M(p_2) = M_0(p_1) + M_0(p_2) = 1$$
$$I_2: \quad M(p_3) + M(p_4) = M_0(p_3) + M_0(p_4) = 1.$$

The place invariant I_1 indicates that, for any marking reachable from M_0, the sum of the tokens in p_1 and p_2 will always be 1. Therefore, in a marking M where $M(p_2) = 1$, the number of tokens in p_1 will be zero. Additionally, it is known that there will never be more than one token in p_1 and p_2. ◇

2.5.2 Repetitive components, transition invariants

Motivation

Consider again in FIG. 2.23, the subnet formed by the transitions c and d along with their input and output places (p_3, p_4, and p_5). Note that firing the sequence $s = cd$ from the initial marking $M = p_3^3$ returns to the same marking. This subnet is represented in FIG. 2.26.a.

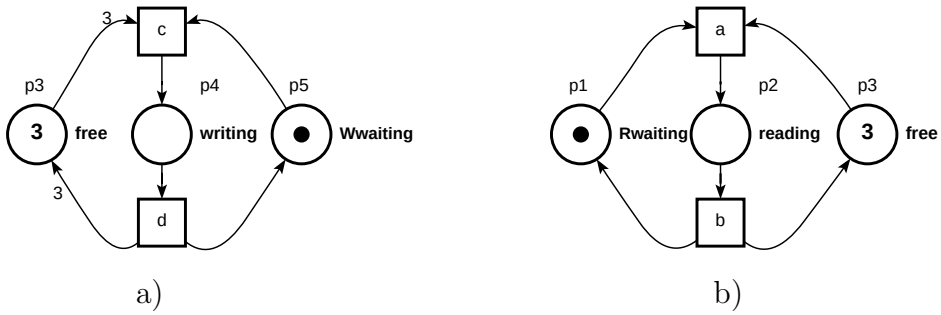


Figure 2.26: Subnets of FIG. 2.23.

The sequence $s = cd$ is a *transition invariant*, since the firing of such a sequence does not change the marking of the net. The transition invariant corresponds to a cyclic sequence of events that can be repeated indefinitely. The set of transitions c and d of the invariant forms a *repetitive component* of the net.

The sub net formed by the transitions a and b along with their input and output places (p_1, p_2, and p_3) is represented in FIG. 2.26.b. The sequence $s = ab$ is another transition invariant of the net in FIG. 2.23.

Definition

The set of transitions that, once fired from a marking of the net, return to the same marking, can be calculated by using the fundamental equation 2.16. It is easy to verify that, to obtain $M' = M$ in equation 2.16, the sequence s must be such that the column vector $\mathbf{s}_{m\times 1}$ satisfies:

$$C\mathbf{s} = 0. \tag{2.30}$$

Every solution $\mathbf{s}$ of this equation is called a *repetitive component*, and the corresponding sequence s is called a *transition invariant* or a *t-invariant*.

A Petri net is said to be *repetitive* if all transitions $t \in T$ belong to a repetitive component. A repetitive component defines a subnet in which only the transitions for which the corresponding component of s is non-zero are considered, along with their input and output places. Other transitions connected to these places are not considered.

The repetitive component does not depend on the initial marking, unlike the corresponding transition invariant, whose existence depends on it.

The integer solutions of Equation 2.30, $C\mathbf{s} = 0$ with $\mathbf{s} > 0$, are also called t-flow. If all the coefficients of a flow are positifs, it is called a t-semi-flow.

If $\mathbf{s}$ is the characteristic vector of a sequence s *effectively* fired from an accessible marking M, then this sequence s is a transition invariant.

The term repetitive may induce an error. It does not mean that the Petri net will actually repeat a sequence of transitions leading back to marking M. Indeed, as discussed in Section 2.1.8, this subnet depicts a (reduced) set of transitions that *could* be fired. It is necessary to check if a combination of these transitions can be fired and lead the Petri net back to marking M. This can be done by successively applying Equation 2.9 or by using a simulator.

Notation:
For the sake of readability, the vector $\mathbf{s}^i$ of repetitive components can be written as a set of transitions $s^i = \{t_j \mid s^i(t_j) \neq 0\}$. The set notation can also be omitted, but it is useful to remember that the repetitive component is an (unordered) set of transitions.

Example 2.25 Repetitive components 1
The repetitive components of the Petri net in FIG. 2.23, obtained applying Equation 2.30 with the matrix C shown in FIG. 2.24, are:

$$\mathbf{s}^1 = [1\ 1\ 0\ 0]^T \qquad \mathbf{s}^2 = [0\ 0\ 1\ 1]^T.$$

They can be written as: $s^1 = \{a, b\}$ and $s^2 = \{c, d\}$. Therefore, the net is repetitive, as all transitions belong to a repetitive component.

They define the sub-Petri nets in FIG 2.26. Applying Equation 2.9 from the initial marking $M = p_3^3$, or using a simulator, reveals that only the sequences $s^1 = ab$ and $s^2 = cd$ are indeed transition invariants. ◇

Example 2.26 Repetitive components 2
Applying the matrix C of the Petri net in FIG 2.21.a (page 62) to equation 2.30 yields the vector $\mathbf{s^1} = [1\ 1\ 1]^T$ which can be written as $s^1 = \{a, b, c\}$. Therefore, the net is repetitive.

In such a small model, it is easy to verify that the sequence $s = bac$ (from home state $M' = p_2p_4$) is indeed a transition invariant and the only one. Any other combination of the set s^1 is not fireable. ◇

Example 2.27 Batch system: structural analysis
Consider the batch-type system from the Example 1.2, whose Petri net is shown in FIG. 1.17.b. The conservative components of this Petri net are given by the vectors $\mathbf{f}^1 = [1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0]^T$, $\mathbf{f}^2 = [0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0]^T$, $\mathbf{f}^3 = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0]^T$ and $\mathbf{f}^4 = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1]^T$, that respectively define the following subnets, represented in FIG. 2.27 formed by:

- places p_1, p_2, p_3, p_4, and p_5, and transitions t_a, t_b, t_c, t_d, t_e, and t_f: this subnet represents the two manufacturing options for product Pr_1;
- places p_6 and p_7, with transitions t_g and t_h: this subnet represents the manufacturing of product Pr_2;
- places p_3 and p_8, with transitions t_b and t_c: this subnet represents the state evolutions of reactor R_1: if occupied, place p_3 is marked; if free, place p_8 is marked;
- places p_5, p_7, and p_9, with transitions t_e, t_f, t_g, and t_h: this subnet represents the state evolutions of reactor R_2 and its possible allocation to one of the two products.

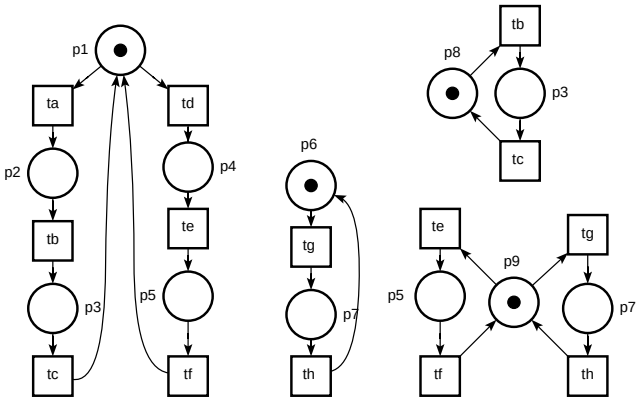


Figure 2.27: Conservative components.

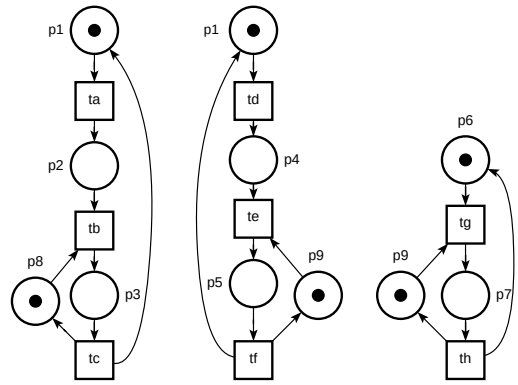


Figure 2.28: Repetitive components.

The Petri net in FIG. 1.17 describes the system as a collection of communicating processes (the subnets defined above) and, in addition to the system's operation, also shows its structure. Therefore, it provides both a behavioral and structural model of the system.

The place invariants I_1 to I_4, obtained from vectors $\mathbf{f}^1$ to $\mathbf{f}^4$ and the initial marking $M_0 = [\,1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\,]^T$ (or $M_0 = p_1p_6p_8p_9$), allow verifying that indeed the system behaves as expected:

- I_1: $M(p_1)+M(p_2)+M(p_3)+M(p_4)+M(p_5) = 1$: the batch Pr_1 can be in waiting (p_1), in B_1 (p_2), in B_2 (p_4), being processed by reactor R_1 (p_3), or reactor R_2 (p_5). The fact that there is only one token in one of the places $p_1 \ldots p_5$ indicates that the system will indeed be in only one of these states.
- I_2: $M(p_6) + M(p_7) = 1$: there is a batch Pr_2 waiting (p_6) or being processed by reactor R_2 (p_7).
- I_3: $M(p_3) + M(p_8) = 1$: reactor R_1 is free (p_8) or processing batch Pr_1 (p_3).
- I_4: $M(p_5)+M(p_7)+M(p_9) = 1$: reactor R_2 is free (p_9), processing batch Pr_1 (p_5), or processing batch Pr_2 (p_7).

The information provided by the invariants is important for a larger Petri net, where it is not always obvious to verify, without calculation, whether two or more places that cannot be marked at the same time are indeed not marked.

The solutions to equation 2.30 are the vectors $\mathbf{s}^1 = [1\ 1\ 1\ 0\ 0\ 0\ 0\ 0]^T$, $\mathbf{s}^2 = [0\ 0\ 0\ 1\ 1\ 1\ 0\ 0]^T$ and $\mathbf{s}^3 = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]^T$. They can be written as $s^1 = \{t_a, t_b, t_c\}$, $s^2 = \{t_d, t_e, t_f\}$, and $s^3 = \{t_g, t_h\}$. These repetitive components form the subnets, represented in FIG. 2.28, with the transitions:

- t_a, t_b, and t_c, with places p_1, p_2, p_3, and p_8: once batch Pr_1 passes through the buffer and is processed by R_1, a new cycle can restart.
- t_d, t_e, and t_f, with places p_1, p_4, p_5, and p_9: same behavior regarding Pr_2 and R_2.
- t_g and t_h, with places p_6, p_7, and p_9: indicates the sequence of events beginning the processing of Pr_2 and its end.

The Petri net is repetitive since $s(t) \neq 0$ for all transitions $t \in T$. By checking all combinations of the transitions for each repetitive component $\mathbf{s}^1$, $\mathbf{s}^2$, and $\mathbf{s}^3$, we find that the sequences $s_1 = t_a t_b t_c$, $s_2 = t_d t_e t_f$, and $s_3 = t_g t_h$ are fireable from initial marking $M_0 = p_1 p_6 p_8 p_9$, and they correspond to realizable t-invariants. ◇

2.6 Notes

AKERKAR (2009) presents knowledge-based systems and a broad variety of knowledge-based techniques for decision support and planning.

2.7 Exercises

1 Review the modeling exercises proposed in Chapter 1. Verify if the parallel and conflicting activities as well as the resources were correctly considered. For each model, find: a) the matrix representation; b) the grammatical representation (alphabet $I\!P$ and rewriting rules Q); c) the set of reachable markings, using equation 2.10; d) the conservative and repetitive components; e) the place and transition invariants.

2 For the net in FIG.2.29, check which sequences are actually fireable. Change the order of some of them (so that the characteristic vector is the same) and verify which ones are not fireable.

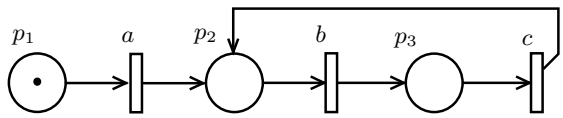


Figure 2.29: Exercises 2 and 3.

3 Check, using the enumeration of markings method, if the net in FIG.2.29 is live, bounded, and reversible. Based on the calculation of repetitive components, verify if the Petri net is repetitive.

4 The Petri net in FIG. 2.30.a describes a robot (free if P_1 is marked) used to load (P_2) and unload (P_5) a milling machine (free if P_4 is marked); the milling operation is associated with place P_3. Determine for this net:
a) The conservative and repetitive components (and their respective place and transition invariants);
b) The set of reachable markings.

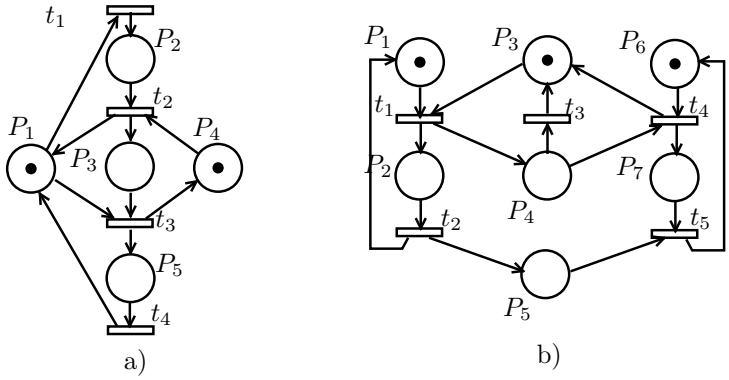


Figure 2.30: Exercise 4 and 5.

5 For the Petri net in FIG. 2.30.b. determine:
a) The conservative and repetitive components (and their respective place and transition invariants);
b) The set of reachable markings.

Chapter 3

Property Analysis

This chapter presents three ways of analyzing the behavior of a Petri net: by marking enumeration, structural analysis and reduction. For the first method, algorithms will be presented that allow us to determine whether a given Petri net has the so-called *good* properties. Although the properties of a live and reversible Petri net have been shown to be decidable, no practical algorithm has been found in the general case. Only the principles of the decidability proof concerning the bounded Petri net property will be given, along with a method for determining other properties. Following this, other methods that partially avoid marking enumeration will be presented.

Structural analysis is performed through the calculation of the conservative and repetitive components and the corresponding invariants. This method, although not always conclusive, allows, in some cases, for simpler and quicker determination of the net's properties.

When it is not possible to obtain the properties of the net through structural analysis, and the net is such that enumerating the markings is too costly, a reduced Petri net can be obtained through the reduction method. The analysis is then conducted on the reduced model.

Finally, the relationships between the different methods will be described, along with the order in which they should be used to obtain the properties of the modeled system.

3.1 Analysis by Marking Enumeration

This method is valid only for the marked Petri net. Therefore, to analyze the behavior of the model for another initial marking, it is necessary to construct a new graph. The advantage of this method is that if it is possible to construct the graph, all the properties can be proved, as will be seen below.

It is initially necessary to observe that it is equivalent to say that the marked Petri net is bounded and that the number of its accessible markings is finite, as the number of places is finite.

In the same way, it is equivalent to say that the set of reachable markings is not finite and that there exists at least one infinite sequence of firings which, when applied, does not lead to any previously calculated marking of the net. Such a sequence is constructed from a finite sequence, but with each firing of this sequence from a marking M, a new marking is reached that is strictly greater than M in at least one place. Therefore, the

set of reachable markings is not finite. One should not confuse an infinite sequence of firings with a finite sequence that repeats indefinitely, always leading, at each firing of this sequence, to the same marking in the set of accessible markings.

3.1.1 Decidability of the k-limited property

When enumerating the reachable markings, if this number is finite, the calculation procedure will stop. However, if the number of markings is infinite, the calculation will never end. Therefore, it is not evident, *a priori*, that the k-limited property is decidable. Decidability arises from the following properties:

1. Monotonicity:

$$\exists M,\ \exists M',\ \exists s \mid M \xrightarrow{s} \text{ and } M' > M \Rightarrow M' \xrightarrow{s} .$$

   If a sequence of transition firings s is fireable from a marking M, it will also be fireable for any marking M' that is greater than M (i.e., it includes all the tokens of M and at least one additional token in some place). It follows from Equation 2.7 that if a transition t is enabled by a marking M, it will also be enabled by M' and can be fired.

2. If $\exists M, M' \in A(R, M) \mid M \xrightarrow{s} M'$ and $M' > M$ then the marked Petri net is unbouded. Indeed, it immediately follows from the previous point that the sequence s can be repeated as many times as desired, and each time, the number of tokens in at least one place is increased because M' is strictly greater than M. The sequence s is a non-stationary infinite-length sequence, which never pass through the same markings.

3. Karp and Miller's Lemma:

   > In any infinite sequence of vectors formed by non-negative integers $v_1, v_2, \ldots, v_k, \ldots$, there exist at least two elements (in fact, an infinite number of pairs) v_i and v_j with $i < j$ such that $v_i < v_j$ or $v_i = v_j$.

   The result is evident for vectors with only one component: an infinite sequence of non-negative integers cannot be strictly decreasing. The proof for the general case is provided in KARP (1969).

4. Now consider an infinite firing sequence s. The markings through which it passes can be written as a sequence of vectors with non-negative integers. The Karp and Miller lemma then implies that there exist at least two markings M_i and M_j in s such that $i < j$ and $M_j > M_i$ or $M_j = M_i$. In the first case, this implies that there exists a subsequence s_1 such that

$$M \xrightarrow{s1} M' \text{ and } M' > M$$

   and the net is unbounded. In the second case ($M_j = M_i$), it means that there is no need to continue, as everything accessible from M_j is also accessible from M_i.

5. From a given marking M, the number of fireable transitions is bounded (the set T is finite), and the set of markings that are immediate successors of M is therefore finite.

3.1.1.1 Coverability Tree

The algorithm that determines if a marked Petri net is k-bounded is based on constructing a tree called the *coverability tree*. Starting from the initial marking M_0, each transition enabled by this marking creates a branch. The markings obtained by firing these transitions are calculated, and the process repeats for each obtained marking. The construction of a branch is interrupted when a marking M is found that:

- Is equal to another already calculated marking for which all successors have already been or will be calculated. The last case corresponds to pending branches that will be explored later (otherwise, a subtree already calculated would be re-explored);
- Is strictly greater than a marking M' *in the branch currently being explored*: $\exists s \mid M_0 \xrightarrow{s} M', \exists s' \mid M' \xrightarrow{s'} M$ and $M > M'$.

In the last case, the exploration of the tree is stopped because the marked net may not be bounded. The value $M'(p)$ that makes the marking strictly greater is denoted by w. Otherwise, the exploration continues until all branches have been explored. This way, it can be asserted with certainty that the marked net is bounded and that the set of reachable markings is known. This procedure characterizes an algorithm because:

- It cannot have any branch of infinite length (Karp and Miller's lemma);
- The number of branches is finite, as for each marking, the number of enabled transitions is finite (less than the number of transitions in the Petri net).

According to the order of the exploration of the branches, whether in depth or in width, the obtained coverability tree can be different for the same Petri net. If the goal is to check if the net is bounded, it may be enough to stop when $M > M'$. However, it can be interesting to continue the calculation to gain more information about the model.

A coverability graph can be obtained from a coverability tree when, instead of interrupting the exploration of the tree when a marking $M'(p) = w$ appears, one continues to calculate the following reachable markings.

The boundedness property, checking that the number of tokens in each place does not exceed a certain limit, ensures that the system does not overflow and that resources are managed efficiently.

Example 3.1 Coverability tree
Consider the Petri net in FIG. 3.1. The coverability tree for this net is depicted on FIG. 3.2. The tree stops at node 2 as $p_1\ p_3 > p_1$ in the branch $0-1-2$. The symbol w is used to indicate the marking of p_3, even if there is only one token for that marking. The same occurs for node 3 ($p_1\ p_2^3 > p_1$ in the branch $0-1-3$). The coverability tree allows to determine that the places p_2 and p_3 are unbouded, and so is the Petri net.

The coverability graph is depicted on FIG. 3.3, continuing the calculation from the coverability tree. This allows us to show that p_1 is also unbouded.

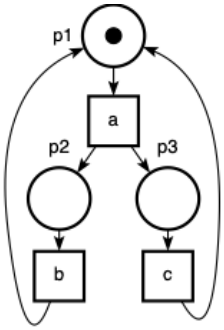
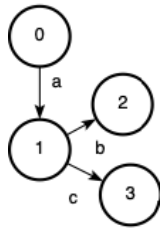
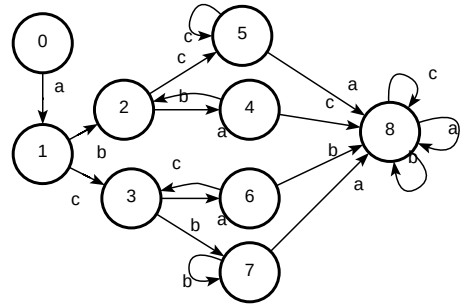


Figure 3.1: Petri net.



$0 : p_1$
$1 : p_2\ p_3$
$2 : p_1\ p_3^w$
$3 : p_1\ p_2^w$

Figure 3.2: Coverability tree.



$4 : p_2\ p_3^w$
$5 : p_1^w\ p_3^w$ $\quad 7 : p_1^w\ p_2^w$
$6 : p_2 w\ p_3$ $\quad 8 : p_1^w\ p_2^w\ p_3^w$

Figure 3.3: Coverability graph.

◇

3.1.2 Searching for Other Properties

It has been shown that the property of boundedness is decidable for a marked Petri net. The construction of the coverability tree allows the verification of this property.

There is no general algorithm for the other properties, although they are, from a theoretical point of view, decidable. However, most of the time, the desired goal is for the constructed net to simultaneously possess all the *good* properties. The first step is to verify if the net is k-bounded and to construct the reachability graph. Then, use graph search algorithms (see Appendix A) to verify if the net is reversible or live.

Reversibility

A marked net $N =< R, M >$ is reversible if and only if its reachability graph $GA(R, M)$ is strongly connected:

$$\forall M_i, M_j \in GA(R, M),\ \exists s \mid M_i \xrightarrow{s} M_j. \tag{3.1}$$

Since the initial marking M is part of the graph, and the graph is strongly connected, there exists a path (sequence s of transitions) between any pair of nodes (markings). In this case, what matters is a path between any marking $M_i \in GA(R, M)$ and the initial marking M.

Liveness

A marked net $N =< R, M >$ is live if the reachability graph $GA(R, M)$ is strongly connected and all transitions $t \in T$ label at least one arc of the graph:

$$\forall t \in T\ \exists M_i,\ \exists M_j \in GA(R, M),\ \mid M_i \xrightarrow{t} M_j. \tag{3.2}$$

Therefore, if the net is reversible, it is enough to prove that it is quasi-live. If every transition $t \in T$ labels an arc (M_i, M_j) (quasi-live net), it is sufficient to verify if there exists a sequence that, after M_j, allows returning to M_i and firing t. Now, if the graph is strongly connected (reversible net), there exists a path between (M_j, M) and (M, M_i), and the net is, therefore, live.

If the graph is not strongly connected (non-reversible net), the net is live if, in each strongly connected pending component (or connected subgraph), all transitions $t \in T$ label at least one arc.

The liveness property ensures that the system can continue to operate indefinitely without getting stuck in a particular state.

Condition 3.2 is sufficient but not not necessary because a Petri net can be live for a given marking without being reversible.

3.1.3 Analysis by Marking Enumeration: Key Takeaways

The analysis approach by enumerating accessible markings (which can be considered verification since all states are generated) is as follows:

1. Verify if the marked net is bounded by constructing the coverability tree.
2. If the marked net is k-bounded, construct the reachability graph by folding the leaves of the tree onto identical markings.
3. Check if the reachability graph is strongly connected.
4. If the reachability graph is strongly connected, verify that every transition appears at least once as a label of an arc in the graph.

This approach allows for verifying if the marked Petri net is simultaneously bounded, reversible, and live.

Consider Example 2.16. As also discussed in Example 2.21, the Petri net shown in FIG.2.15 is not reversible because the graph is not strongly connected. Transitions a, b, c, e, and f are live as they appear in both strongly connected pending components (subgraphs). Transition d is quasi-live as it does not appear in any of the subgraphs. herefore, the net is not live (it is quasi-live).

On the other hand, the net in Fig.2.21.a (Exemple 2.20), whose reachability graph is given in Fig.2.21.b, is live because within the strongly connected pending component formed by markings $M = p_2p_4$, $M = p_1p_3$ and $M = p_2p_3$, all transitions in the net (a, b, and c) label an arc.

3.2 Structural Analysis

The structural analysis method, based on repetitive and conservative components, as discussed in Section 2.5, provides information about the behavior of the unmarked net. The advantage is that the results are valid for any marking. However, it is not possible to obtain conclusive information when $\mathbf{f}(p) = 0$ and when $s(t) \neq 0$.

Place and transition invariants, derived from conservative and repetitive components respectively, depend on the initial marking. Nevertheless, their calculation is simple and fast, requiring only the substitution of the M_0 value in Equation 2.29 and writing the transition sequences from the repetitive components. However, the same observation applies: the result is not always conclusive.

3.2.1 Conservative Components, Place Invariants

Conservative components allow us to consider only the structure of the Petri net, that is, the unmarked net. The following information is obtained regarding the property of boundedness:

- Any place that belongs to a conservative component is bounded;
- Even if a place p does not belong to any conservative component ($\mathbf{f}(p) = 0$), p can still be bounded; therefore, a non-conservative net can be bounded;
- An unbounded place does not belong to any conservative component; therefore, an unbounded net is non-conservative;
- A conservative net is bounded (for any marking) since all its places are bounded.

Although it may seem strange at first that a property apparently so tied to the marking can be obtained solely from the structure of the net, remember that the matrix C in Equation 2.28 updates the tokens in the net, indicating the number of tokens added to and removed from each place.

From the perspective of token conservation in the net, structural analysis provides the following two results:

Place Coverage × Boundedness:

A Petri net for which there exists a coverage of conservative components[1] $\mathbf{f}^T > 0$ is k-bounded *regardless of its initial marking.*

Place Invariants × Limit Value:

The linear form $\mathbf{f}^T M = \mathbf{f}^T M_0$ allows to calculate a limit for each place. Indeed, as the components of $\mathbf{f}$ and the markings are non-negative numbers, we have for any place p:

$$\mathbf{f}(p)M(p) \leq \mathbf{f}^T M_0$$

and consequently, the following limit (which is not necessarily achieved):

$$M(p) \leq \frac{\mathbf{f}^T M_0}{\mathbf{f}(p)}. \tag{3.3}$$

Example 3.2 p-invariant information

As seen in Example 2.24,the place invariants of the net in FIG.2.21.a, for the initial marking $M_0 = [\,0\ 1\ 0\ 1\,]^T$ are given by:

$$\begin{aligned} I_1 :&\quad M(p_1) + M(p_2) = 1 \\ I_2 :&\quad M(p_3) + M(p_4) = 1. \end{aligned}$$

Invariant I_1 shows that places p_1 and p_2 are binary, because for any marking reachable from M_0, the maximum number of tokens is 1. Another piece of information provided by this invariant is that when place p_1 is marked, the number of tokens in p_2 is zero (the same reasoning applies to invariant I_2 and places p_3 and p_4). This information goes

[1] A *coverage* of conservative components (respectively, repetitive) is formed by a *positive* component — or a set of *positive* elementary components — that spans all places (respectively transitions).

beyond the notion of boundedness: it shows the relationship that exists between the markings of places. To establish this relationship among all places in the net, simply calculate the linear combination $\mathbf{f} = \mathbf{f}^1 + \mathbf{f}^2 = [\,1\ 1\ 1\ 1\,]^T$, and find the corresponding linear place invariant (for $M_0 = [\,0\ 1\ 0\ 1\,]^T$):

$$M(p_1) + M(p_2) + M(p_3) + M(p_4) = 2.$$

The net is conservative because all places belong to a conservative component. ◇

Example 3.3 Non-conservative Petri nets
Consider Example 2.15. The Petri net depicted in FIG. 2.12 is non-conservative as it has no semiflows, as well the net in FIG. 2.19. Then nothing can be said about their boundness. However the coverability graphs allows to proof that the Petri net in FIG. 2.12 is unbounded whereas the one in FIG. 2.19 is bounded.

The Petri net shown in FIG. 2.13 is also non-conservative: it has one conservative component: $f = p_1p_2$ but p_3 does not belong to any conservative component. It proves that p_1 and p_2 are bounded but there is no proof for the p_3 boundness.

The Petri net shown in FIG. 2.22, Example 2.21, whose matrix C is given by:

$$C = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

has two conservative components: $\mathbf{f}^1 = [\,1\ 1\ 0\ 0\ 0\,]$ and $\mathbf{f}^2 = [\,0\ 0\ 0\ 1\ 1\,]$. There is no coverage of place, since for any linear combination of $\mathbf{f}^1$ and $\mathbf{f}^2$, $\mathbf{f}(p_3) = 0$. Therefore, the net is non-conservative. However, this net is live and binary for the initial marking $M = p_1p_3$, which is easily verifiable by calculating the reachable markings. For the initial marking $M = p_1p_3p_4$, the Petri net is unbounded. Therefore, it is not possible to conclude about the token limit in a place p for which $\mathbf{f}(p) = 0$. ◇

Example 3.4 Batch system: conservative components
Consider the batch-type system presented in Example 1.2, whose conservative components were calculated in Example 2.27.

The support of the conservative components is given by $\mathbf{f} = \mathbf{f}^1 + \mathbf{f}^2 + \mathbf{f}^3 + \mathbf{f}^4 = [\,1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\,]^T$. Since all places belong to the support (indeed, $\forall p \in P, \mathbf{f}(p) \neq 0$), the net is conservative. Therefore, without calculating the reachable markings graph, the structural analysis proves that all places are bounded, and that the net is bounded. It is not necessary to know the initial marking of the net; this result is valid regardless of the initial marking.

The invariants I_1 to I_4 (Example 2.27) allow calculating an upper bound for the marking of each place. This bound may never occur; what is guaranteed is that the marking will always be less than or equal to it. The calculation of the invariants is valid only for the initial marking used in Equation 2.29. From invariant I_1, for $M_0 = [\,1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\,]^T$, we obtain the equation:

$$\forall M \in A(R, M), \quad M(p_1) + M(p_2) + M(p_3) + M(p_4) + M(p_5) = 1$$

and it can be verified that, for any reachable marking, the maximum number of tokens, for places p_1, p_2, p_3, p_4, and p_5, is one. Again, it is known that the limit is one, but it cannot be guaranteed that it will be reached, that is, whether these places will actually be marked for any marking. Analyzing the other invariants, it can be verified that the net is safe, as the maximum number of tokens for all places in the Petri net is one. ◇

It is necessary, however, to emphasize that a Petri net can be bounded for a given marking without having a coverage of conservative components. This is the case for the example in the following.

3.2.2 Repetitive Components, Transition Invariants

Liveness and boundedness properties can also be defined from repetitive components:

- Every live, bounded, and reversible Petri net is repetitive (the converse is false).
- If the net is non-repetitive ($\exists t,\ s(t) = 0$) then either it is not live or it is not bounded.
- A repetitive net ($\forall t,\ s(t) > 0$) may not be live.

Transition Coverage × Bounded and Live:
Every Petri net that is both live and bounded for at least one initial marking has a coverage of repetitive components $\overline{s} > 0$. This follows from the following facts:

- Bounded ⇒ there is a finite number of markings enabling a given transition;
- Live ⇒ there are infinite-length sequences;
- Bounded and live ⇒ there is a repetitive sequence.

It's straightforward to confirm that the opposite of the second point is incorrect. Typically, one can select a marking in which all places are empty.

The Petri net in FIG. 2.21.a is bounded, live, and reversible for the marking $M = p_2p_4$, as seen in the graph in FIG.2.21.b (Example 2.20), using the analysis by enumeration of markings. Therefore, the net must be repetitive, as indeed it is, since $\mathbf{s} = [1\ 1\ 1]^T$, as seen in Example 2.26. However, the opposite is not true, as this repetitive net is not reversible for the marking $M = p_1p_4$, although it is bounded and live.

Example 3.5 Non-repetitive Petri nets
Consider again Example 2.15. The incidence matrix C of the Petri net in FIG. 2.12 is given by:

$$C = \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}.$$

The only value of $\mathbf{s}$ that satisfies Equation 2.30 is $\mathbf{s} = [0\ 0\ 0]^T$, which indicates that the net is non-repetitive. Consequently, it is either not live or not bounded. By constructing the reachable markings graph, it can be verified that the net is indeed unbounded.

The net in FIG. 2.19 (Example 2.19) has one repetitive component, $s = t_1 t_2$. However, transition t_3 does not appear in a repetitive component, meaning that either t_3 is not live or the net is unbounded. Since the net is non-conservative, as seen in Example 3.3, structural analysis does not provide useful information. The marking graph proves that this Petri net is bounded and reversible but not live, as t_3 is not live. ◇

Example 3.6 Repetitive Petri nets
The Example 2.27 has presented the repetitive components of the batch-type system. The support of repetitive components is given by $\mathbf{s} = \mathbf{s}^1 + \mathbf{s}^2 + \mathbf{s}^3 = [\,1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\,]^T$. Since all transitions belong to the support (indeed, $\forall t \in T, s(t) \neq 0$), the net is repetitive.

The net in FIG. 2.9 is also repetitive, with $\mathbf{s} = [\,1\ 1\ 1\ 1\,]^T$ but is not live for the marking $M = p_1 p_2$, as seen in Example 2.10. It is, therefore, important to note that the existence of a coverage of repetitive components does not guarantee liveness.

The Petri net shown in FIG. 2.13 has two repetitive components, $\mathbf{s}^1 = t_2$ and $\mathbf{s}^2 = t_1 t_3 t_4$. All transitions appear in a repetitive component therefore the net is repetitive. All these three models are repetitive, but nothing can be said about their liveness. ◇

Why does an element $s(t) = 0$ imply that the net is either not live or not bounded? The idea behind the repetitive component, and in particular, the corresponding transition invariant, is that firing this sequence does not change the initial marking, as indicated by equation 2.30. Now, if a transition t does not belong to a repetitive component, then it does not belong to any sequence that leads back to the initial marking. There are two reasons for this:

- The net is unbounded, it is impossible to return to the initial marking because there is a growth in tokens with each firing of t;
- The net is not live because transition t has a deadlock, making it impossible to return to the initial marking.

Why does an element $s(t) \neq 0$ not allow us to determine if the net is live? This fact stems from the limitation of matrix C to represent the Petri net[2], as extensively discussed in Section 2.1.8.2.

Although the structural analysis of the net does not always provide a conclusive answer about the properties of the model, its computation is straightforward, without the time and memory costs required for computing the reachable markings graph. Moreover, it provides, at a glance, information about the structure, such as the token limit for a set of places or the set of transitions whose firing would lead to the initial marking.

3.3 Analysis through Reduction

The method of analysis by enumeration of markings, presented in Section 3.1, is extremely burdensome to implement when the size of the set of reachable markings becomes large, due to the combinatorial explosion of the number of states. One solution is to apply reduction rules so that the initial Petri net and the reduced net are equivalent in terms

[2]See chapter 8, where linear logic is used to characterize the firing sequence.

of properties. Thus, the enumeration analysis method can be applied to a reasonably sized net, and the properties of the original net can be deduced.

The notion of equivalence does not necessarily imply that the two nets have the same sequences of transition firings or the same place or transition invariants. The goal is that the general properties, if verified in the reduced net, to be considered as those of the original net. However, in certain specific cases, the application of some rules necessary to reduce the net does not preserve all good properties.

3.3.1 Substitutable Place

Informally, a substitutable (or removable or deletable) place is a place that serves solely as an intermediate step between two transitions (or sets of transitions). The firing of the input transition (or one of the input transitions) is a sufficient condition to enable the firing of the output transition (or one of the output transitions). The simplification involves merging the input transitions with the output transitions, two by two.

Let p be a place with initial marking $M(p)$ and let t_o be an output transition from p and t_i an input transition to p. For p to be substitutable, it is necessary that:

$$\begin{aligned} \forall t_i \in T, \forall t_o \in T,\ & Pre(p, t_o) = Post(p, t_i) \quad \text{and} \\ \forall q \neq p \in P, \forall t_o \in T,\ & Pre(q, t_o) = 0 \qquad (3.4) \\ & M(p) = 0. \qquad (3.5) \end{aligned}$$

and the pair $(t_i\ t_o)$ is replaced by the transition t_{io} such that:

$$\begin{aligned} \forall q \in P, q \neq p,\ Pre(q, t_{io}) &= Pre(q, t_i) \quad \text{and} \\ Post(q, t_{io}) &= Post(q, t_i) + Post(q, t_o). \qquad (3.6) \end{aligned}$$

thus eliminating place p.

Example 3.7 Substitutable place
Consider the Petri net in FIG .3.4. The firing of one of the two transitions a or b is a sufficient condition to enable the firing of transition c or d (both are in conflict). Place p_4 serves only as a passage and can be removed. Its removal yields the Petri net in FIG.3.5.

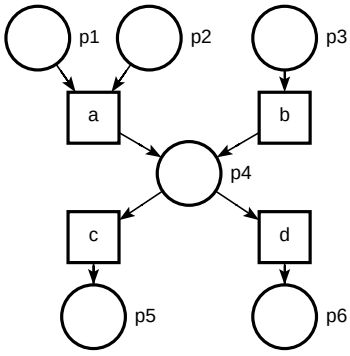


Figure 3.4: Replaceable place p_4.

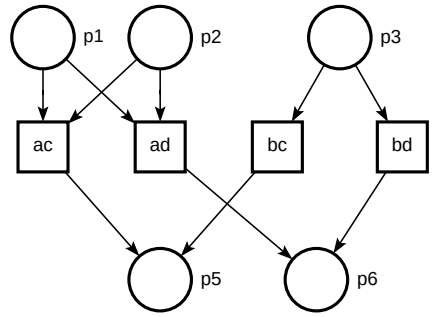


Figure 3.5: Place p_4 removed.

◇

The input transitions t_i of a place p can have other output places besides p, which is not an issue. For example, in FIG. 3.6, transition t_1 has place p_1 as an output in the original model, and p_1 remains an output place of transition t_{12} in the reduced Petri net.

Figure 3.6: a) Replaceable place p_2; b) Place p_2 removed.

The most common case corresponds to long linear sequences (places and transitions have only one incoming arc and one outgoing arc), as in the Petri net in FIG. 3.7. The sequences will be condensed into a model with a single transition, as in the case of t_{abcd}, depicted in FIG. 3.8.

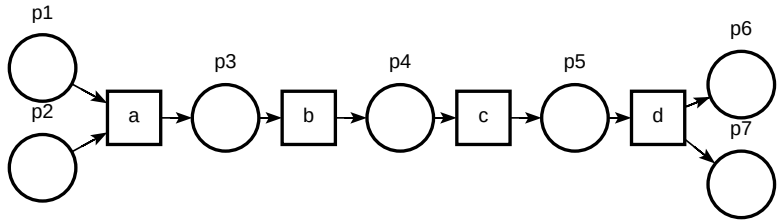


Figure 3.7: Substitutable places p_3, p_4, p_5.

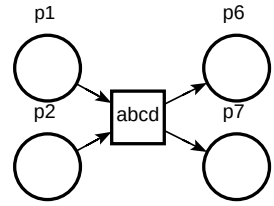


Figure 3.8: Reduced PN.

Two important remarks should be highlighted regarding the preservation of Petri net properties:

1. If the substitutable place contains tokens, it is necessary to fire the output transition before removing it. However, in this case, the reduced net and the original net are no longer equivalent in terms of the reversibility;

2. If the output transition of the substitutable place does not have output places, the reduced net and the original net are no longer equivalent in terms of the property of k-boundness.

Example 3.8 Non-preservation of properties
Refer back to Example 2.20. To make place p_1 of the net shown in FIG. 2.21 substitutable, transition a must be fired. The new initial marking in the reduced net is p_2p_4, which makes it reversible, whereas the original Petri net was not reversible for the marking p_1p_4.

Now take place p_2 from the Petri net in FIG. 3.6.a, which is substitutable. This place is unbounded because it is possible to fire transition t_1 without firing transition t_2 as many times as desired. However, the reduced net in FIG. 3.6.b becomes bounded.

If transition t_2 had an output place in the original net, the reduced net would remain unbounded as well. ◇

3.3.2 Implicit Place

In a way, a substitutable place is not very useful since it is simply a relay between two transitions. Similarly, we can say that an implicit place is unnecessary because its marking should be calculable from the marking of a subset of places and it should not introduce any additional constraint for the firing of its output transitions. Of course, implicit places generally have a clear meaning in terms of the system being modeled.

3.3.2.1 General case:

If the marking of a place $p \in P$ is a linear combination of the marking of a set of other places in the Petri net $E \subset P$, this means that there is a conservative component coverage $\{p\} \cup E$. We therefore have the following definition.

A place p of a Petri net $< P, T, Pre, Post >$ whose initial marking is M_0 is implicit if and only if it belongs to a conservative component $\mathbf{f}$ covering a set $\{p\} \cup E$ such that:

$$\mathbf{f}(p) < 0 \quad \text{and } \forall p_i \in E, \mathbf{f}(p_i) > 0 \tag{3.7}$$

$$M_0(p) \geq \frac{\sum_i \mathbf{f}(p_i).M_0(p_i)}{\mathbf{f}(p)} \tag{3.8}$$

$$\forall t \in T \quad \text{and } Pre(p,t) \neq 0; \exists p_i \in E \mid Pre(p,t) \leq \frac{\mathbf{f}(p_i)}{\mathbf{f}(p)} \,.\, Pre(p_i,t) \tag{3.9}$$

The condition 3.7 indicates that the conservative component $\mathbf{f}$ must have a negative weight for p and positive weights for the other places p_i. Indeed, the place invariant $\mathbf{f} \cdot M = \mathbf{f} \cdot M_0$ must show that there are sufficient tokens in p.

The condition 3.8 is a condition on the initial marking. If there are initially tokens in places p_i but none in place p, p might have an insufficient marking and thus limit the firing of its output transitions.

The condition 3.9 implies that p does not limit the firing of t any more than the places p_i do, considering the weights of the places in the conservative component $\mathbf{f}$.

The reachability graph of the Petri net $\langle P, T, Pre, Post \rangle$ is the same as that of the reduced Petri net obtained by eliminating place p. Therefore, all the good properties (k-bounded, live, and reversible) are preserved.

Example 3.9 Implicit place
Consider the Petri net in FIG. 3.9.a. The aim is to show that place p_1 is implicit with respect to the set of places $E = \{p_2, p_3\}$, and can therefore be removed. To do so, one must search for the conservative components.

The basis B of linearly independent vectors is formed by the conservative components $\mathbf{f}^1 = [0\ 1\ 1\ 1\ 1]^T$ and $\mathbf{f}^2 = [1\ 0\ 0\ 1\ 1]^T$. The goal here is to find a vector $\mathbf{f}$, which is a combination of vectors $\mathbf{f}^1$ and $\mathbf{f}^2$, covering the set $p_1 \cup E = \{p_2, p_3\}$ such that its components f_4 and f_5 are zero. Such a vector is obtained through the linear combination $\mathbf{f} = \mathbf{f}^1 - \mathbf{f}^2 = [-1\ 1\ 1\ 0\ 0]^T$, with $f(p_1) = -1$ which satisfies condition 3.7.

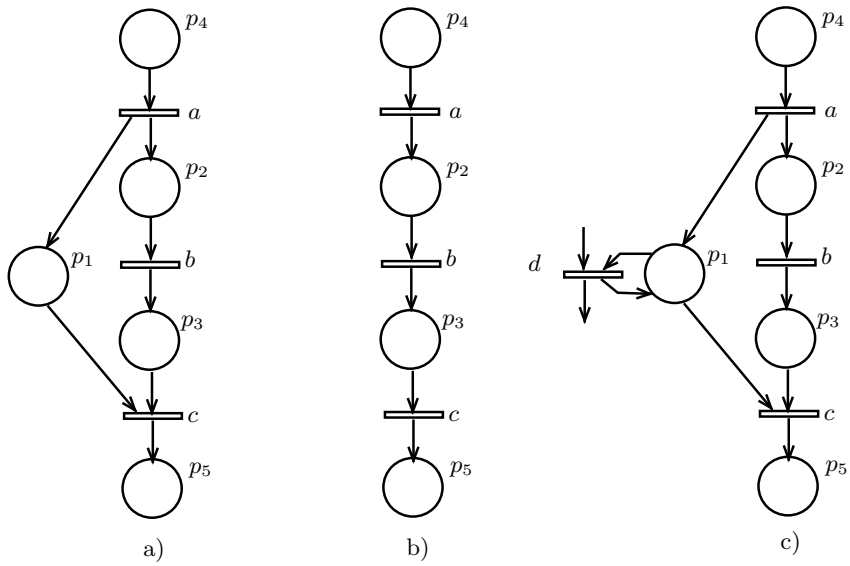


Figure 3.9: a) Implicit place p_1. b) Reduced net. c) Counterexample.

Applying $\mathbf{f}$ in equation 2.29, with the initial marking $M_0 = [\,0\ 0\ 0\ 0\ 0\,]^T$, yields the linear place invariant:

$$M(1) = M(2) + M(3)$$

Therefore, one can calculate at any instant the marking of place p_1 as a function of the markings of places p_2 and p_3. This linear invariant satisfies condition 3.8:
$M_0(p_1) = \frac{\mathbf{f}(p_2).M_0(p_2)+\mathbf{f}(p_3).M_0(p_3)}{\mathbf{f}(p_1)} = 0$.

The only output transition from place p_1 is transition c, which satisfies condition 3.9:
$Pre(p_1, t) = {}^{\mathbf{f}(p_3)}/_{\mathbf{f}(p_1)}\,.\,Pre(p_3, c) = Pre(p_3, c)$.

Therefore, place p_1 can be removed without affecting the firing sequences of the transitions. Thus, we obtain the reduced net shown in FIG. 3.9.b. ◇

Naturally, from the perspective of representing system states, a substitutable place may play an important role. For instance, the marked place p_1 in Example 3.9 indicates that the process formed by the activities executed between the firing of a and c is still ongoing.

Two key factors need to be considered.

- The initial marking of the net alters the equation used to calculate the marking of the implicit place.
- It is not sufficient for the marking of a place to be a function of the marking of a subset E for it to be implicit.

These issues will be addressed in the following examples.

Example 3.10 Implicit place (cont.)
Continuing from Example 3.9, if place p_2 (or p_3) is marked, $M_0' = [\,0\ 1\ 0\ 0\ 0\,]^T$ (or $M_0' = [\,0\ 0\ 1\ 0\ 0\,]^T$), the new invariant is:

$$M(1) = M(2) + M(3) - 1.$$

Consequently, the fact that place p_2 contains a token is no longer sufficient to assert that place p_1 contains at least one token. However, for the initial marking $M'_0 = [\,1\ 0\ 0\ 0\ 0\,]^T$ (only place p_1 is marked), the new invariant is $M(1) = M(2) + M(3) + 1$, and p_1 is an implicit place. ◇

Example 3.11 Implicit place counterexample
Consider the Petri net model in FIG. 3.9.c. Transition d is an output transition from place p_1, which has at least one input place (not shown) that does not belong to the set $E = \{p_2, p_3\}$: it does not satisfies condition 3.8. In this case, place p_1 is not implicit. Indeed, the fact that the other input place of d has a token does not imply that p_1 also has a token. Moreover, in this net, the firing of d can only occur after the firing of a (and before the firing of c). If place p_1 is removed, this condition for the firing of d no longer exists, which alters the behavior of the net. ◇

3.3.2.2 Identical Places and Degenerated Implicit Places:

Two types of implicit places, illustrated by FIG. 3.10.a and 3.10.b, are easily recognized: identical places and degenerated implicit places.

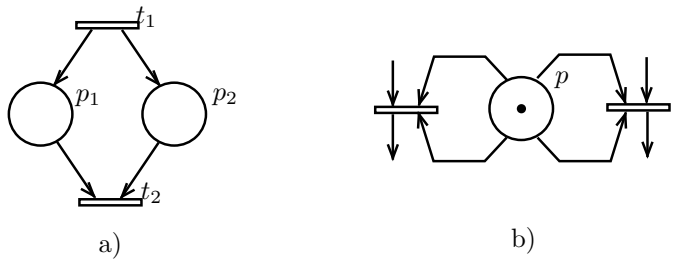


Figure 3.10: a) Identical places; b) Degenerated implicit place.

- Two places p_i and p_j are identical if they have the same input and output transitions, with the same arc weights:

$$\begin{aligned} \forall t \in T\,,\ Pre(p_i,t) &= Pre(p_j,t),\ Post(p_i,t) = Post(p_j,t) \ \text{ and} \\ M_0(p_i) &= M_0(p_j) \end{aligned} \tag{3.10}$$

  This corresponds to the case where the set E is reduced to a single place, $\{p_j\}$). We then have $M(p_i) = M(p_j)$. This is the case for places p_1 and p_2 in FIG. 3.10.a. It is then said that p_1 is implicit in relation to p_2 and can be removed.

- The place p is a degenerate implicit place if it is connected to the Petri net only by elementary loops and its initial marking is sufficient:

$$\forall t \in T\,,\ Pre(p,t) = Post(p,t) \ \text{ and} \tag{3.11}$$
$$\forall t \in T\,,\ M_0(p) \geq Pre(p,t) \tag{3.12}$$

  This corresponds to the case where the set E is empty. The marking of place p does not change. This is the case for places p in FIG. 3.10.b.

The same observation made for the Petri net in FIG. 3.9.b applies to the net in FIG. 3.10.b: the result of the analysis does not change when removing place p, but in this case, the information that the resource p is needed and shared by the two transitions is lost.

3.3.3 Identity Transition

The transition t_i in a Petri net $\langle P, T, Pre, Post\rangle$ is an *identity* transition, or a neutral transition, if it is connected to the rest of the Petri net only by elementary loops:

$$\forall p \in P\,,\ Pre(p, t_i) = Post(p, t_i), \tag{3.13}$$

and one of these conditions is satisfied:

$$\exists t_j \in T\,, \forall p \in P\,,\ Pre(p, t_i) \leq Pre(p, t_j) \tag{3.14}$$

$$\text{or} \qquad \exists t_k \in T\,, \forall p \in P\,,\ Pre(p, t_i) \leq Post(p, t_k). \tag{3.15}$$

The net is reduced by removing the transition t_i, which does not change the liveness of the Petri net.

The firing of t_i does not change the marking (condition 3.13), and either there exists a transition t_j that, if it is enabled from a marking M_i, ensures that t_i was also enabled (condition 3.14), or there exists a transition t_k that produces a marking M_k allowing t_i to fire (condition 3.15).

In other words, condition 3.14 guarantees that t_j produces a marking that enables t_i, and condition 3.15 guarantees that t_k has the same firing conditions as t_i. In both cases, if these transitions are live in the reduced net, it ensures that t_i is live in the initial net. Removing t_i does not change the liveness of the Petri net. Otherwise, what may be being removed from the net is a dead part of it. The reduced net may, in this case, be live, while the original net is not, as shown in Example 3.13.

Example 3.12 Identity transitions
An identity transition is illustrated in FIG. 3.11.a by transition t. Transition t_1 satisfies condition 3.14 and transition t_2 satisfies condition 3.15. It is sufficient for one of the conditions to be fulfilled.
In the Petri net of FIG. 2.17, described in Example 2.17, transition g is neutral (it satisfies condition 3.13)) and can be removed, as transition d has the same firing conditions (it satisfies condition 3.14). The original and the reduced net (depicted in FIG. 2.16) are both quasi-live. ◇

Example 3.13 Identity transition counterexample
A counterexample is provided by the Petri net in FIG. 3.11.b. Although transition d satisfies Equation 3.13, it cannot be removed because neither condition 3.14 nor condition 3.15 is fulfilled. The original net is quasi-live, as shown in the marking graph in FIG. 3.11.c: after the firing of transition a, transition d can no longer be fired. Conversely, the reduced net, obtained by suppressing d, is live, as depicted in FIG. 3.11.d. ◇

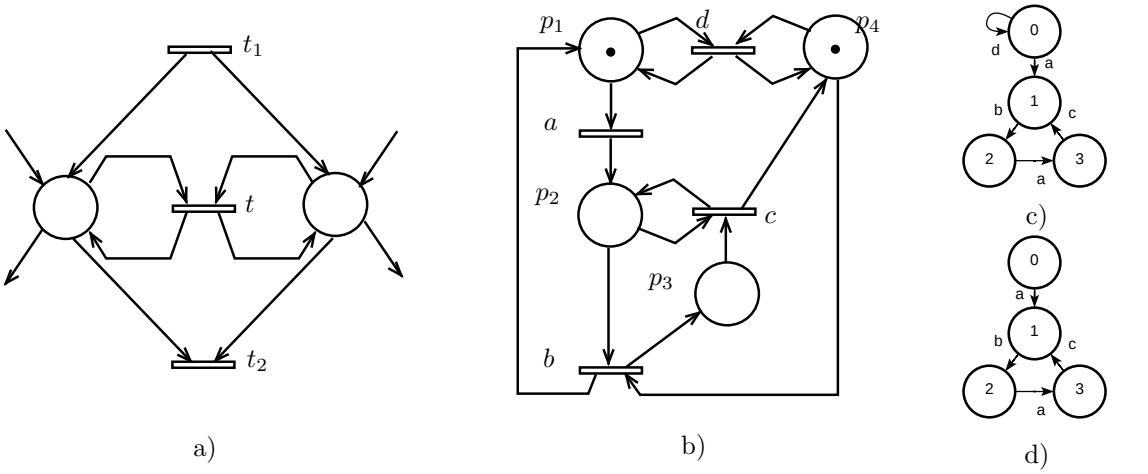


Figure 3.11: a) Neutral transition t; b) d (not simplifiable); c) Marking graph of FIG. 3.11.b; d) Marking graph of FIG. 3.11.b removing transition d.

3.3.4 Identical Transitions

Two transitions t_i and t_j in a Petri net $\langle P, T, Pre, Post\rangle$ are identical if they have the same input and output places with the same arc weights:

$$\forall p \in P\,,\ Pre(p,t_i) = Pre(p,t_j) \ \text{ and } \ Post(p,t_i) = Post(p,t_j)$$

It is evident that one of the two can be eliminated without changing the properties of the Petri net.

Example 3.14 Identical transitions
The Petri net in FIG. 3.12.a provides an example of two identical transitions t_1 and t_2; the simplified net with respect to transition t_2 is shown in FIG.3.12.b.

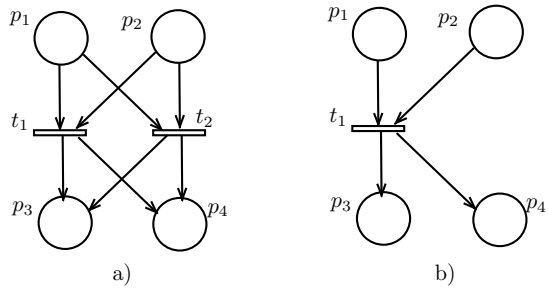


Figure 3.12: a) Identical transitions t_1 and t_2; b) Simplification of t_2

◇

Example 3.15 Batch system: analysis through reduction
Consider again the
batch-type system from the Example 1.2 and its Petri net model shown in FIG. 1.17.

Place p_3 is substitutable because conditions 3.4 and 3.5 are satisfied. Its output transition (t_c) has an output place (p_1), so the properties will be preserved in the reduced

net (see remark page 3.3.1). Places p_5 and p_7 are substitutable for the same reasons. The reduced net is shown in FIG. 3.13.

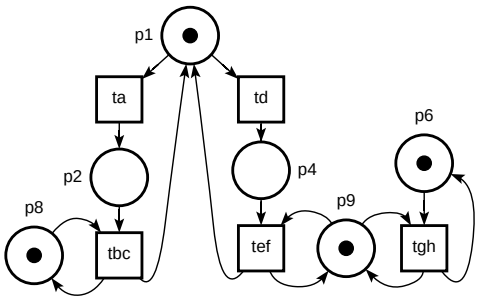


Figure 3.13: Reduced Petri net, first step.

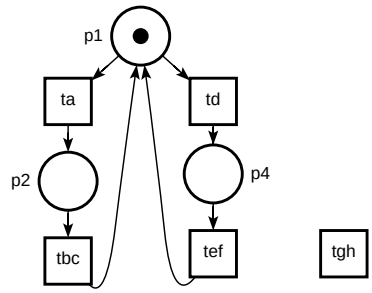


Figure 3.14: Reduced Petri net, 2nd step.

Places p_6, p_8, and p_9 are implicit degenerate places because condition 3.11 is satisfied. From condition 3.12, we have: $M_0(p_6) \geq Pre(p_6, t_{gh}) = 1$, $M_0(p_8) \geq 1$ and $M_0(p_9) \geq 1$. The reduced net is shown in FIG. 3.14.

Places p_2 and p_4 in the net shown in FIG. 3.14 are substitutable because conditions 3.4 and 3.5 are satisfied. Their output transitions (t_{bc} and t_{ef}) have an output place (p_1), so the properties will be preserved in the reduced net shown in FIG. 3.15.

In the third step, place p_1 becomes an implicit degenerate place and can be eliminated, provided it contains at least one token (condition 3.12 issatisfied for $M_0(p_1) \geq 1$). The reduced net is shown in FIG. 3.16.

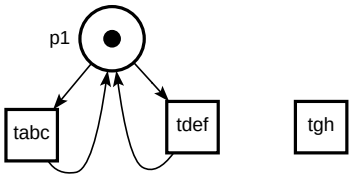


Figure 3.15: Reduced Petri net, 3rd step.

Figure 3.16: Reduced Petri net, 4th step.

Finally, in this last reduction step, transitions t_{gh} and t_{def} are identical to transition t_{abc}. The net is reduced to a single transition and no places. It is k-bounded because no place can contain an infinite number of tokens, and it is live and reversible since transition t_{abc} can be fired without constraint.

Note that this is a degenerate case of a Petri net since there are no places. However, it can still be represented by a system of rewriting rules. The initial marking is the empty set $M_0 = \emptyset$, and the transition is $t_{abc} : \emptyset \rightarrow \emptyset$.

This matches the result of the marking enumeration analysis (FIG. 1.18), indicating that the Petri net is k-bounded, live, and reversible for its initial marking. However, it is important to emphasize that these properties are verified for all initial markings of the form:

$$M_0 = [\, n_1\, 0\, 0\, 0\, 0\, n_6\, 0\, n_8\, n_9\,]^T$$

with

$$n_1 \geq 1,\ n_6 \geq 1,\ n_8 \geq 1,\ n_9 \geq 1$$

and that it is therefore possible to *increase the number of batches to be processed* (p_1 and p_6) and the number of reactors of each t ype R_1 (p_8) and R_2 (p_9) without issue.

However, it is clear that the reduced Petri net has lost all significance relative to the modeled system. ◇

3.4 Relationship between various analysis methods

A distinction is sometimes made between *verification* and *formal proof*. Verification involves enumerating all possible cases before concluding whether a property is true or not. Formal proof is a deductive process that allows conclusions to be drawn without enumerating all cases. Therefore, the analysis by enumerating the markings accessible from a given initial marking falls under verification, while the calculation of components is more akin to proof. However, if the net is not k-bounded, only the coverability tree is constructed, which may be considered a proof. When the net is reduced before enumerating the markings, the method falls somewhere between proof and verification. Although the calculation of components is more of a proof, enumerating all components is a verification process. Thus, the distinction is far from clear, and we prefer to speak of analysis.

When dealing with a Petri net representing a somewhat complex system, no method guarantees that the properties of a marked Petri net can be fully obtained. Marking enumeration (Section 3.1) can be difficult due to a combinatorial explosion in the number of states. More importantly, it does not allow one to determine whether a non-k-bounded net is live. Yet, analyzing the behavior of a system with a few unbounded intermediate stocks might be necessary. Additionally, this approach must be entirely redone whenever the initial marking changes. However, the initial marking of some places may represent a pool of available resources, and it would be beneficial to have a result independent of the exact number of resources.

It may also happen that it is not necessary or interesting to construct the set of reachable markings. Therefore, one must resort to one of the other two methods to obtain information about the system: structural analysis and reduction.

Considering the reduction method (Section 3.3), it is only possible if its structure is not too complex. The use of the *substitutable place* rule may require changing the initial marking. However, the elimination of a degenerate implicit place, imposing only a minimum value for the marking, allows for considering cases with undefined initial markings. For example, in the Petri net shown in Figure 3.10.b, place p can be removed for any initial marking $M(p) \geq 1$.

Moreover, if the reduced net lacks the desired properties, the cause of the problem remains unknown, with no clear path to correction. If the net is not k-bounded, the specific places that are unbounded remain unidentified. If the net is not live, the unexecutable transitions or the markings corresponding to deadlock remain unknown. The reduction procedure gradually diminishes the significance of places and transitions, making it impossible to relate them to the elements of the system being modeled.

Structural analysis (Section 3.2), through the calculation of positive conservative and repetitive components, provides useful information for correcting an incorrect model. Places not belonging to any positive conservative component are likely to be unbounded. Transitions not belonging to any repetitive component are likely to be non-live, or the net can be unbounded.

As we have seen, structural analysis essentially decomposes the initial net into p-

subnets and t-subnets, which generally have a clear meaning concerning the system being modeled. *p-subnets* typically correspond to *resources passing through various states*, while *t-subnets* describe behaviors, or *sequences of events* involving various resources.

The use of place invariants allows us to show that certain places are bounded and calculate their bound, without enumerating the set of reachable markings. Transition invariants only provide necessary but not sufficient conditions. However, they provide a set of transitions whose appropriate order in a sequence could lead back to the initial marking.

In conclusion, we see that it is often useful to combine various approaches: reducing the initial net, enumerating the markings of the reduced net, identifying components to better understand the model, etc.

Characterization of markings

Let us consider a marked Petri net $\langle R, M_0 \rangle$, where $R = \langle P, T, Pre, Post \rangle$ with incidence matrix $C = Post - Pre$.

Several sets of markings can be defined. First, there is the set of markings that are actually reachable from M_0, denoted as $A(R, M_0)$, by firing the transitions of R, using Equation 2.9.

One can also define the set of positive (or zero) markings M, which are solutions to the characteristic equation $M = M_0 + C.\mathbf{s}$ (Equation 2.16), where $\mathbf{s}$ is any vector of non-negative integers corresponding to different firing sequences. It is possible to write a vector $\mathbf{s} > 0$ that does not correspond to any fireable sequence s, as discussed in Section 2.1.8. However, if the resulting marking M is negative, such a sequence is not fireable.

Finally, from the set of conservative components $\{\mathbf{f}^i\}$, solutions to $\mathbf{f}^T.C = 0$ (Equation 2.28), one can consider all the non-negative solutions to $\mathbf{f}^{iT}.M = \mathbf{f}^{iT}.M_0$ (Equation 2.29), which represent all the markings compatible with the place invariants of R for M_0.

It is evident that reachable markings are solutions to the characteristic equation and that the solutions to the characteristic equation are compatible with the conservative components.

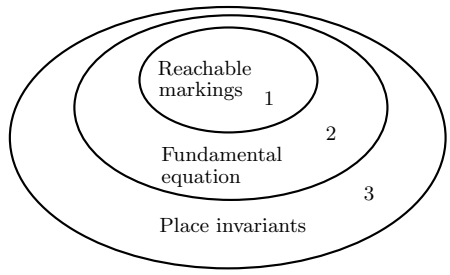


Figure 3.17: Characterization of markings.

The question that arises is: are these three sets identical? The answer is no. FIG. 3.17 illustrates the relationship between the different sets of markings obtained in each of the three sets. Set "1" corresponds to the reachable markings $A(R, M_0)$. Set "2" corresponds to the positive markings M', which are solutions to the fundamental equation for $\mathbf{s} \geq 0$.

This set necessarily includes the first one but may be larger. Set "3", defined by the linear place invariants, includes set "2" but may be strictly larger.

These different ways of characterizing the markings can be illustrated through the following example.

Example 3.16 Marking characterization
Consider the Petri net in FIG. 3.18, with $M_0 = [\,1\ 0\ 0\ 0\,]^T$, whose matrix C is shown in FIG. 3.19.

The set of reachable markings, obtained through the application of Equation 2.9, is given by $\{M_0, M_1\}$, with $M_0 \xrightarrow{a} M_1 = [\,0\ 1\ 0\ 0\,]^T$. It corresponds to set "1" in FIG.3.17.

To apply Equation 2.16 from the initial marking M_0, it is necessary to list all possible (positive) characteristic vector $\mathbf{s}$, as seen in Section 2.1.8.1.

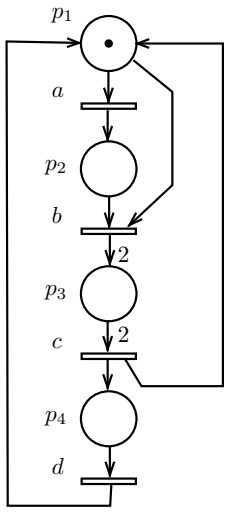


Figure 3.18: Different sets of markings.

$$C = \begin{bmatrix} -1 & -1 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 2 & -2 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

Figure 3.19: Matrix C of FIG 3.18.

Vectors $\mathbf{s}^1 = [1\ 0\ 0\ 0]^T$, $\mathbf{s}^2 = [1\ 1\ 1\ 0]^T$, and $\mathbf{s}^3 = [1\ 1\ 1\ 1]^T$, correspond respectively to sequences $s_1 = a$, $s_2 = abc$, and $s_3 = abcd$ (as well as their permutations). These vectors lead to positive markings when applying Equation 2.16 from $M_0 = [\,1\ 0\ 0\ 0\,]^T$. Specificaly, $\mathbf{s}^1$ results in $M_1 = [\,0\ 1\ 0\ 0\,]^T$, $\mathbf{s}^2$ yelds $M_2 = [\,0\ 0\ 0\ 1\,]^T$, and $\mathbf{s}^3$ brings the marking back to $M_3 = [\,1\ 0\ 0\ 0\,]^T = M_0$.

The vector $\mathbf{s}^4 = [1\ 1\ 0\ 0]^T$ (associated with sequences ab and ba) would produce the solution $M_4 = [-1\ 0\ 2\ 0]^T$, which is not acceptable because it is a negative marking. Other vectors as $[0\ 1\ 0\ 0]^T$, $[0\ 0\ 1\ 0]^T$, and $[0\ 0\ 0\ 1]^T$, produce also negative markings.

Other possible vectors do not produce any new markings. Thus, the set of markings, obtained through the application of Equation 2.16, is given by $\{M_0, M_1, M_2\}$. It corresponds to set "2" in FIG.3.17. However, M_2 is not a reachable marking from M_0, as can be seen by moving the tokens in FIG. 3.18.

Let us consider now the place invariants. The Petri net in FIG. 3.18 has a single conservative component $\mathbf{f} = [\,1\ 1\ 1\ 1\,]^T$ (see calculation method in Appendix A). Applying Equation 2.29 for the marking $M_0 = [\,1\ 0\ 0\ 0\,]^T$, the linear place invariant obtained is:

$$M(p_1) + M(p_2) + M(p_3) + M(p_4) = 1.$$

The only markings that satisfy this invariant are as follows: $M_0 = p_1$, $M_1 = p_2$, $M_2 = p_4$, and $M_3 = p_3$, because the sum of the markings of all places must be equal to 1. The set of markings $\{M_0, M_1, M_2, M_3\}$ corresponds to set "3" in FIG. 3.17. Markings M_0 and M_1 are the only truly accessible ones; marking M_2, present in set "2", and marking M_3 are not reachable from M_0. ◇

This example illustrates that the only way to truly determine the set of reachable markings is to calculate the graph of reachable markings by successively applying equation 2.9. This is why the enumeration of markings is sometimes indispensable. When the three sets are equal (or very similar), structural analysis using positive conservative and repetitive components is the most comprehensive.

3.5 Particular Results: subclasses

This section discusses three subclasses of Petri nets with unitary weighting for which there are specific analysis results. For other Petri nets with unitary weighting (beyond these subclasses), it's necessary to use the methods seen previously.

3.5.1 Binary Petri net and Grafcet

A binary Petri net is a 1-bounded Petri net. The marking of places is either zero or one token. Thus, places can be equated with logical conditions, true (one token) or false (zero tokens). This class corresponds to sequential logical controllers (and therefore, also to Grafcet). It is the only class for which simple implementations can be realized. The coverability tree is simpler because the algorithm stops as soon as a place contains more than one token. Therefore, the method of analyzing reachable markings is often preferred.

3.5.2 State Machines

A subclass of Petri net with unitary weighting is a state machine (not to be confused with the concept of finite state machines) if and only if very transition has exactly one input place and exactly one output place.

Formally, a Petri net $R = \langle P, T, Pre, Post\rangle$ is a state machine if:

$$\forall t \in T,\ \exists p_i \in P \mid Pre(p_i, t) = 1,\ \exists p_j \in P,\ Post(p_j, t) = 1 \tag{3.16}$$

$$\text{and}\ \ \forall p \in P,\ p \neq p_i, Pre(p, t) = 0,\ \forall p \in P,\ p \neq p_j,\ Post(p, t) = 0. \tag{3.17}$$

This subclass models conflict but does not model either concurrency or synchronization.

A Petri net of the state machine type forms a conservative component that encompasses all places. Therefore, it is bounded regardless of its initial marking, and all properties are decidable. Since transitions have only one input place, a Petri net of the state machine type is live and reversible if it contains at least one token in each repetitive component, and if each of these components is strongly connected.

Example 3.17 State machine
The two Petri nets in FIG. 3.20 belong to the state machine subclass because they satisfy

conditions 3.16 and 3.17. Therefore, both are bounded. The repetitive components $\mathbf{s}^1 = [1\ 1\ 0\ 0]^T$ and $\mathbf{s}^2 = [0\ 0\ 1\ 1]^T$ of the Petri net on the left are shown in FIG. 3.21. Since each component is strongly connected and contains at least one token (in this case, three tokens), then the Petri nets in FIG. 3.20 are also live and reversible.

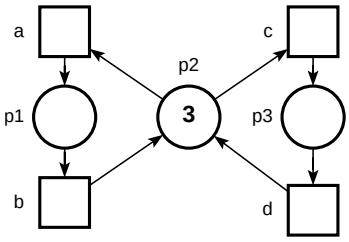
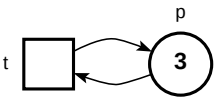


Figure 3.20: Petri nets of state machine subclass.



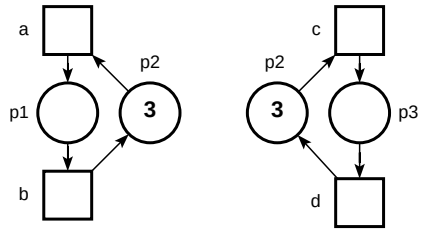


Figure 3.21: Repetitive components.

Notice that the Petri net in FIG. 2.1 is not a state machine since it does not satisfies condition 3.16 because $Pre(p_2, c) = 3$ and $Post(p_2, d) = 3$. ◇

3.5.3 Event Graph

A subclass of Petri nets with unit weighting is an event graph if and only if very place has exactly one input transition and exactly one output transition.

Formally, a Petri net $< P, T, Pre, Post >$ is a state machine if:

$$\forall p \in P, \quad \exists t_i \in T,\ Pre(p, t_i) = 1,\ \forall t \in T,\ t \neq t_i,\ \ Pre(p, t) = 0 \qquad (3.18)$$

$$\text{and} \qquad \exists t_j \in T,\ Post(p, t_j) = 1,\ \forall t \in T,\ t \neq t_j\ \ Post(p, t) = 0. \qquad (3.19)$$

A circuit in an event graph is a non-null length path, starting from t_{j_1} and returning to t_{j_1}. It is the graphical representation of a sequence of transitions $s = t_{j_1} t_{j_2} \ldots t_{j_k}$ such that $t_k = t_{j_1}$ and:

$$\forall (t_{j_r}, t_{j_{r+1}})\ \exists p_{i_r} \mid p_{i_r} \in Post(t_{j_r}) \text{ and } p_{i_r} \in Pre(t_{j_{r+1}}).$$

The circuit s is elementary (it does not pass through the same transition twice, except for the first and last) if:

$$\forall t_k \in s\,, \forall t_l \in s \text{ with } t_k \neq t_{j_1}\,,\ t_l \neq t_{j_k} \ \text{ then } \ t_k \neq t_l.$$

Every elementary circuit of an event graph is a conservative component.

The number of tokens in a circuit of the event graph does not change with firings. As a consequence, these are necessary and sufficient conditions:

- An event graph is live for a given initial marking if and only each of its elementary circuits contains at least one token. A live event graph is reversible.
- An event graph is bounded (regardless of its initial marking) if and only if every place belongs to at least one of its elementary circuits. If every place that belongs to a circuit contains only one token the event graph is safe.

Therefore, liveness and boundedness properties are decidable in a Petri net of the event graph type.

Example 3.18 Event graph
The Petri net in FIG. 3.22.a is an *event graph* because every place has only one incoming and one outgoing arc. The three elementary circuits C_1, C_2, and C_3, are shown in FIG. 3.22.b. They correspond exactly to the conservative components calculated using Equation 2.28: $\mathbf{f}^1 = [1\ 1\ 0\ 0\ 0\ 0]$, $\mathbf{f}^2 = [0\ 1\ 1\ 1\ 1\ 0]$ and $\mathbf{f}^3 = [0\ 0\ 0\ 0\ 1\ 1]$.

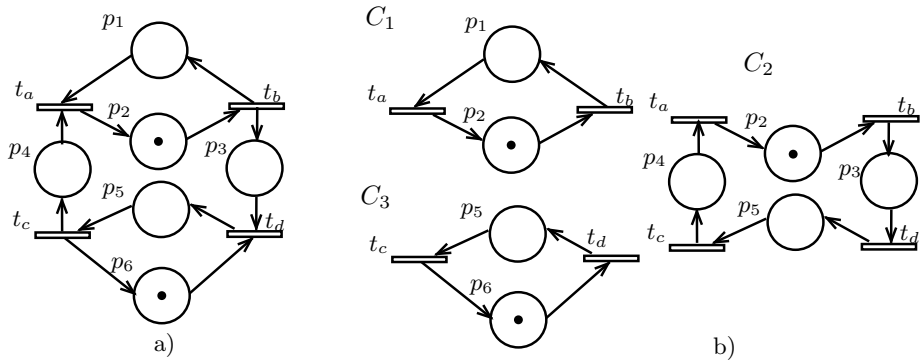


Figure 3.22: a) Event graph; b) Elementary circuits.

In all elementary circuits in FIG. 3.22.b, there is one token; therefore, the Petri net shown in FIG. 3.22.a is live. All places belong to some elementary circuit; thus, this Petri net is bounded. It is safe, as all places in a circuit contain only one token. ◇

3.6 Validation and Verification Using Petri Nets

In the context of system modeling and analysis, validation and verification are critical processes that ensure the system behaves as intended and meets its requirements. Petri nets, as a powerful formal modeling tool, play a significant role in both these processes.

Verification is the process of checking whether a system conforms to its specification. Using Petri nets for verification involves analyzing the structural and behavioral properties of the modeled system to ensure it meets predefined criteria. Key aspects of verification with Petri nets include the proof of liveness, boundedness and reversibility properties. Other properties can be proved using model checking CLARKE (1999) such as:

- Reachability Analysis: Determining whether a certain state (marking) is reached, or can be reached from the initial state. This helps in identifying deadlocks, livelocks, and other critical states.

- Safety Properties: Verifying that certain undesirable states are never reached, ensuring the system avoids critical failures.

Validation, on the other hand, is the process of ensuring that the system fulfills its intended purpose and meets the needs of its users. Validation using Petri nets

involves simulating the model to observe and analyze its dynamic behavior under various conditions. Key aspects of validation with Petri nets include:

- Simulation: Running the Petri net model with different scenarios and inputs to observe its behavior. This helps in understanding how the system operates in practice and identifying potential issues.
- Behavioral Analysis: Examining the sequences of events and transitions to ensure they align with the expected operational procedures and goals.

Verification provides confidence that the system is mathematically and logically sound. This reduces the risk of critical failures and ensures robust system performance. Validation ensures that the system performs as expected in real-world scenarios, meeting the practical needs and expectations of its users.

The following simple example can show the difference and the complementarity of this dual approach.

Example 3.19 Validation vs. Verification
Paul and Mary want to design a robot. Paul needs a robot that emits light and plays a sound (at the same time) when it starts an operation. Mary needs a robot that goes right or goes left. The designer made the Petri nets in FIG. 3.23 to FIG. 3.25.

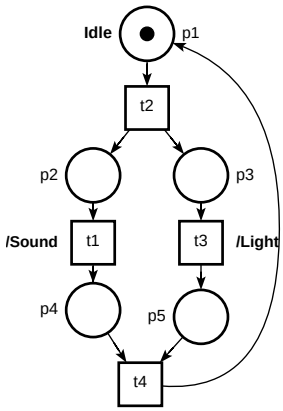


Figure 3.23: PN1.

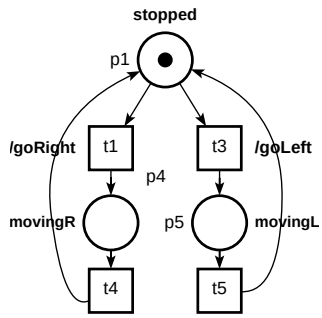


Figure 3.24: PN2.

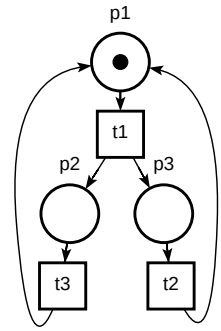


Figure 3.25: PN3.

The verification consists of answering the following question: Do these nets have the *good* properties (bounded, live, reversible)? Constructing the marking graph allows for proving that the Petri nets in FIG. 3.23 and FIG. 3.24 are both bounded, live, and reversible. Therefore, both models are sound. The one in FIG. 3.25, as seen in Example 3.1, is unbouded. This model is not correct. Indeed, transition t_1 is fork, but the join is missing.

As for the validation, two scenarios can allow to check the user specification:

- Scenario 1: the robot starts the operation, in parallel it plays a sound and emits light, then it ends the operation.
- Scenario 2: the robot goes right, move for a while; goes back to the initial point, then goes right again and goes back, then goes left and goes back to the initial point.

Scenario 1 allows for checking Paul's specification. It can be executed in the Petri net of FIG. 3.23 by firing the sequence $s_1 = t_2t_1t_3t_4$, where t_2 corresponds to "start the operation", t_1 to "play a song", t_3 to "emits light", and t_4 to "end of operation".

Scenario 2 allows for checking Mary's specification. It can be executed in the Petri net of FIG. 3.24 by firing the sequence $s_2 = t_1t_4t_1t_4t_3t_4$, where t_1 corresponds to "go right", t_3 to "go left", t_4 to "go back from the right", and t_5 to "go back from the left".

The model PN2, even if it has all the good properties, is a wrong model for Paul, in the same way that the model PN1 is a wrong model for Mary. ◇

Key Takeaways:
The following methods have been seen in this chapter for analyzing the Petri nets:

- Marking graph: Provides *proof* about liveness and reversibility *when* the Petri net is bounded.
- Coverability tree: allows us to determine *which places* can be unbounded, but provides no information about liveness and reversibility.
- Conservative component: A place p is bounded if it belongs to at least one conservative component. If p does not belong to any conservative component, nothing can be said about its boundedness.
- Repetitive components: If a transition t does not belong to any repetitive component, it is not live or it leads to an unbounded marking. If t belongs to some repetitive component, nothing can be said about its liveness. Only the construction of the coverability graph can prove that p_4 is unbounded.

All these methods allow for the verification of a Petri net and is a necessary step in the design for ensuring correctness. Indeed, in general, if the net does not have good properties, it is because the modeling is incorrect and should be reviewed.

However, verification is not sufficient and is essential to also validate the model. Structural analysis can provide information about the model's structure, even if it is not possible to proof the liveness and boundness. You can check with a simulator if a t-invariant leads back to initial marking M_0, and you can use the p-invariant for ensuring some user specification. Validation by simulation is an important and user-friendly step in the design process. However, some bad scenario can be missed in the simulation, as in the Petri net of FIG. 3.31 (try to fire the sequence of transitions $t_{a1}t_{b1}$). It is crucial not only to build expected scenarios, but also unexpected and bad ones.

3.7 Some techniques for modeling large systems

When modeling large systems, the key issues of a Petri net representation are the following:

- Calculability: Is there any algorithm for the analysis?
- Modularity: Is it possible to decompose a complex system?

- Composition: If the modules have the good properties, does the composition of such modules also have good properties? Or is it necessary to analyze the overall (composed) system?

The first question was answered by the algorithms used for the formal analysis of a Petri net model presented in previous sections. This section shows how to obtain a (overall) Petri net model of a complex system from smaller Petri net models, using the concepts of stepwise refinement and composition.

3.7.1 Stepwise Refinement

Stepwise refinement is a *top-down* design where a transition in a Petri net can be replaced by a entire Petri net. It is possible to start with a small overall model, and refine it with more details each time a transition is replaced. To begin with a correct model, a well-formed block with all the necessary properties is first defined.

Definition 3.7.1 (Well-formed block)**.** *A well-formed block is a Petri net with one input transition t_i and one output transition t_o. If a place p is added to the well-formed block such that $Pre(p, t_i) = Post(p, t_o) = 1$, then the Petri net is bounded, live, and reversible.*

The well-formed block (WFB) is represented inside the red dotted line in FIG. 3.26. Examples of WFB are: sequence, if-then-else, do-while, fork-join, etc.

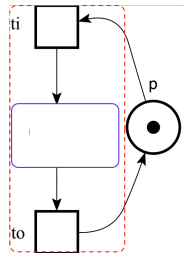


Figure 3.26: A well-formed block connected to a place p.

Stepwise refinement is a process of abstraction done in two steps:

- Start with a rough draft model providing an abstract vision of the overall system, with *shortened* transitions (associated with complex tasks).
- From this initial Petri net, replace the *shortened* transitions with *well-formed blocks* representing a more detailed vision of these complex tasks.

If the abstract Petri net is a well-formed block and the transitions are replaced by well-formed blocks, then the obtained overall Petri net is well-formed and is also bounded, live and reversible. This stepwise refinement approach makes it possible to model and analyze complex systems, while preserving the properties of individual, well-formed blocks.

3.7.2 Composition

Petri net composition is an *object-oriented* process and allows for a *Botton-up* design. The system to be modeled is decomposed in objects or modules. Composition is also done in two steps:

- Start from a *detailed* Petri net model of each module.
- The overall Petri net is constructed from the composition of these models, by merging some of the individual Petri nets's places or transitions.

Unlike stepwise refinement, even if each module is bounded, live and reversible, the overall Petri net obtained by composition does not necessarily preserve these properties. Therefore, it is necessary to analyze the obtained overall model by one of the three methods seen in this chapter. When using analysis by marking enumeration, or through reduction, there is no relation between the analysis of the overall net and the original nets. It is not the case when using structural analysis, as it will be seen in the upcoming sections on Petri net composition.

Even though it is necessary to analyze the composed net, it is mandatory to analyze the Petri net of each module model before the composition.

Let us recall the key points about Petri net structural properties and analysis, as discussed in Sections 2.5 and 3.2. Conservative components, which are solutions to the equation $\mathbf{f}^T M = 0$, and repetitive components, which solve the equation $C\mathbf{s} = 0$, depend solely on the structure of the Petri net and are independent of the initial marking. Place invariants (or p-invariants) and transition invariants (or t-invariants), however, do depend on the initial marking. A repetitive component indicates a *possible loop* in the Petri net. A *t-invariant*, or *realizable t-invariant*, if it exists for a repetitive component, is an ordered sequence of transitions that *actually* that can be repetitively fired from a reachable marking. The term *realizable* emphasizes this distinction.

3.7.2.1 Synchronous composition

In a synchronous composition, transitions that represent the same events in both modules are merged.

Let $R_a =< P_a, T_a, Pre_a, Post_a >$ and $R_b =< P_b, T_b, Pre_b, Post_b >$ be two Petri nets. Let T_c be the set of transitions associated with the same events in R_a and R_b. These are the transitions that will be merged to build the overall Petri net. Define $T_{ai} = T_a \setminus T_c$ as the set of internal transitions of R_a, and similarly, $T_{bi} = T_b \setminus T_c$ as the set of internal transitions of R_b.

We denote $Pre_a = [Pre_{ai}\ Pre_{ac}]$ and $Pre_b = [Pre_{bc}\ Pre_{bi}]$, with the same notation applying for $Post_a$ and $Post_b$.

The Petri net from merging R_a and R_b is the net

$$R =< P_a \cup P_b, T_{ai} \cup T_c \cup T_{bi}, Pre, Post >$$

with

$$Pre = \begin{array}{c} \begin{matrix} T_{ai} & T_c & T_{bi} \end{matrix} \\ \begin{bmatrix} Pre_{ai} & Pre_{ac} & 0 \\ 0 & Pre_{bc} & Pre_{bi} \end{bmatrix} \end{array} \begin{matrix} P_a \\ P_b \end{matrix}$$

and

$$Post = \begin{array}{c} \begin{array}{ccc} T_{ai} & T_c & T_{bi} \end{array} \\ \left[\begin{array}{ccc} Post_{ai} & Post_{ac} & 0 \\ 0 & Post_{bc} & Post_{bi} \end{array}\right] \end{array} \begin{array}{c} P_a \\ P_b \end{array}.$$

Let M_a be the initial marking of R_a and M_b the initial marking of R_b. The initial marking of the overall net R is given by $M^T = [M_a^T \;\; M_b^T]$.

As previously mentioned, the overall Petri net resulting from the synchronous composition of live, bounded, and reversible nets is not necessarily live, bounded or reversible.

However, as directly derived from the construction of Pre and $Post$ matrix, conservative components are preserved in a synchronous composition, and new overall conservative components can emerge.

Internal repetitive components that involve only the transitions of T_{ai} or T_{bi} are also preserved, while new overall repetitive components can emerge.

Proof. Let us note $C_a = [C_{ai}\ C_{ac}]$ and $C_b = [C_{bc}\ C_{bi}]$. The incidence matrix of the overall Petri net is then :

$$C = Post - Pre = \begin{array}{c} \begin{array}{ccc} T_{ai} & T_c & T_{bi} \end{array} \\ \left[\begin{array}{ccc} C_{ai} & C_{ac} & 0 \\ 0 & C_{bc} & C_{bi} \end{array}\right] \end{array} \begin{array}{c} P_a \\ P_b \end{array}$$

With $\mathbf{f}^T$ written $[\ \mathbf{f}_a^T\ \mathbf{f}_b^T\ ]$, the equation $\mathbf{f}^T.C = 0$ becomes $[\ \mathbf{f}_a^T\ \mathbf{f}_b^T\ ].C$ which can be developed into :

$$[\ [\mathbf{f}_a^T.C_{ai}]\ \ [\mathbf{f}_a^T.C_{ac} + \mathbf{f}_b^T.C_{bc}]\ \ [\mathbf{f}_b^T.C_{bi}]\ ] = [\ 0\ \ 0\ \ 0\ ]$$

By choosing $\mathbf{f}_b = 0$ or $\mathbf{f}_a = 0$ it is clear that we find the conservative components of the subnets R_a and R_b.

In the same way, the equation $C.\mathbf{s} = 0$ produces

$$[\ [C_{ai}.\mathbf{s}_{ai} + C_{ac}.\mathbf{s}_c]\ \ \ [C_{bc}.\mathbf{s}_c + C_{bi}.\mathbf{s}_{bi}]\ ]^T = [0\ \ 0]^T$$

By choosing $\mathbf{s}_c = 0$ and $\mathbf{s}_{bi} = 0$ we obtain the equations defining the *internal* repetitive components of R_a. By choosing $\mathbf{s}_{ai} = 0$ and $\mathbf{s}_c = 0$ we obtain the equations defining the *internal* repetitive components of R_b. They are therefore preserved. □

Example 3.20 Synchronous composition

Consider the Petri net modules R_a and R_b represented on the left and right, respectively, in FIG. 3.27. Both are bounded, live, and reversible. R_a has one conservative component $\mathbf{f}_a^T = [1\ 1]$ (or $f_a = \{p_1, p_2\}$) and one repetitive component $\mathbf{s}_a^T = [1\ 1]$ (or $s_a = \{t_1, t_2\}$). R_b has one conservative component $\mathbf{f}_b^T = [1\ 1]$ (or $f_b = \{p_3, p_4\}$) and one repetitive component $\mathbf{s}_b^T = [1\ 1]$ (or $s_b = \{t_3, t_4\}$). For the initial marking $M_a^T = [1\ 0]$ in R_a, the sequence $s_a = t_1t_2$ is a realizable t-invariant, as is the sequence $s_b = t_4t_3$ for $M_b^T = [0\ 1]$ in R_b.

FIG. 3.28 depicts the synchronous composition of these nets, obtained by merging transitions t_1 and t_3 to form transition t_{13}, and merging t_2 and t_4 to form transition t_{24}.

In other words, t_1 and t_3 are assumed to correspond to the same event and are both renamed t_{13}. Similarly, transitions t_2 and t_4 are renamed t_{24} and the set T_c is $\{t_{13}, t_{24}\}$, with T_{ai} and T_{bi} being empty in this elementary case.

The overall net has four conservative components: $\mathbf{f}_1^T = [\mathbf{f}_a^T \ 0\ ]$, $\mathbf{f}_2^T = [0 \ \ \mathbf{f}_b^T]$, which retain the preserved conservative components $\mathbf{f}_a$ and $\mathbf{f}_b$, and two new ones, $\mathbf{f}_3^T : [1\ 0\ 0\ 1]$ (or $f_3 = \{p_1, p_4\}$), and $\mathbf{f}_4^T : [0\ 1\ 1\ 0]$ (or $f_4 = \{p_2, p_3\}$). The overall net is proved to be bounded. There is a new repetitive component $\mathbf{s}^T = [1\ 1]$ (or $s = \{t_{13}, t_{24}\}$), so nothing can be deduced about liveness, and marking enumeration must be performed. In fact, the resulting Petri net in FIG. 3.28 is not live, as neither t_{13} nor t_{24} can be fired from the initial marking $M^T = [1\ 0\ 0\ 1]$ (or $M = p_1p_4$). The vector $\mathbf{s}$ does not correspond to any realizable t-invariant for this initial marking.

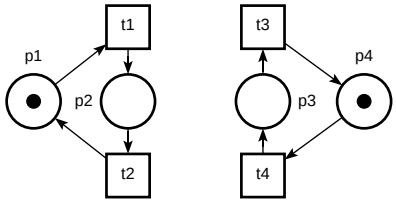


Figure 3.27: Two Petri net modules.

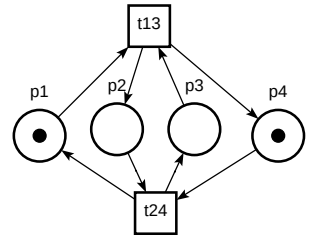


Figure 3.28: Synchronous composition.

One reason the overall net is not live could be due to the choice of the initial marking of R_b. If it were set to p_3 instead of p_4, the new initial marking of the overall Petri net R would be p_1p_3. It retains the preserved internal conservative components $\mathbf{f}_a$ and $\mathbf{f}_b$ and has two new ones, $f_3 = \{p_1, p_2\}$ and $f_4 = \{p_3, p_4\}$. The net is proved to be bounded. The new repetitive component $\{t_{13}, t_{24}\}$, for initial marking $M = p_1p_3$, does correspond to the realizable t-invariant $s = t_{13}t_{24}$.

Another reason could be a design error because in fact the transitions to be merged, i.e., representing the same event, are t_1 and t_4 (transition t_{14}) and t_2 and t_3 (transition t_{23}). The obtained Petri net in FIG. 3.29 is then live, bounded and reversible for initial marking $M = p_1p_4$.

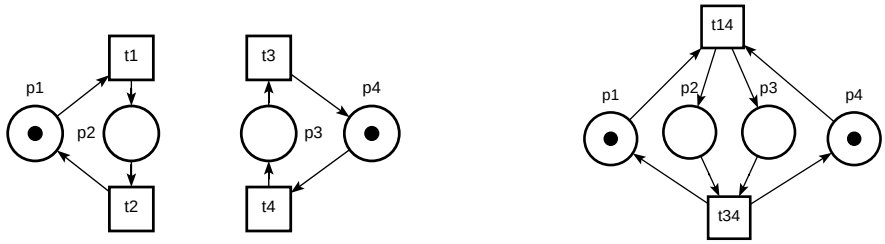


Figure 3.29: Synchronous composition of $t_1 - t_4$ and $t_2 - t_3$.

Only the designer, with knowledge of the system, can determine where the error occurred.

◇

An example is provided in Appendix D, demonstrating how an itinerary of the transportation system from Example 1.1 is derived through the synchronous composition of sections. A further example is presented in Section C.2.

3.7.2.2 Asynchronous composition

In an asynchronous composition places that represent the same activities (or the same resource states) in both modules are merged. This approach can be used to build a system sharing resources. The places representing the idle state of a shared resource involved in various contexts are merged.

Let $R_a =< P_a, T_a, Pre_a, Post_a >$ and $R_b =< P_b, T_b, Pre_b, Post_b >$ be two Petri nets. Let P_c be the set of places associated with the same activities or resource states in R_a and R_b – these are the places that will be merged to build the overall Petri net. They have therefore the same initial marking in R_a and R_b. Define $P_{ai} = P_a \setminus P_c$ as the set of internal places of R_a, and similarly, $P_{bi} = P_b \setminus P_c$ as the set of internal places of R_b.

We denote $Pre_a = [Pre_{ai}\ Pre_{ac}]^T$ and $Pre_b = [Pre_{bc}\ Pre_{bi}]^T$, with the same notation applying for $Post_a$ and $Post_b$.

The Petri net from merging R_a and R_b is the net

$$R =< P_{ai} \cup P_c \cup P_{bi}, T_a \cup T_b, Pre, Post >$$

with

$$Pre = \begin{array}{c} \begin{matrix} T_a & T_b \end{matrix} \\ \left[\begin{array}{cc} Pre_{ai} & 0 \\ Pre_{ac} & Pre_{bc} \\ 0 & Pre_{bi} \end{array}\right] \end{array} \begin{matrix} P_{ai} \\ P_c \\ P_{bi} \end{matrix}$$

and

$$Post = \begin{array}{c} \begin{matrix} T_a & T_b \end{matrix} \\ \left[\begin{array}{cc} Post_{ai} & 0 \\ Post_{ac} & Post_{bc} \\ 0 & Post_{bi} \end{array}\right] \end{array} \begin{matrix} P_{ai} \\ P_c \\ P_{bi} \end{matrix}$$

As in the case of synchronous composition, the overall Petri net obtained by the asynchronous composition of live, bounded, and reversible nets is not necessarily live, bounded and/or reversible.

However, as directly derived from the construction of Pre and $Post$ matrix, repetitive components are preserved in an asynchronous composition, and new overall repetitive components can emerge.

Internal conservative components involving only the places of P_{ai} or P_{bi} are also preserved, while new conservative components can emerge.

Example 3.21 Asynchronous composition
Consider the Petri net modules shown in FIG. 3.30. Two resources, P_{c1} and P_{c2} (e.g., robots), are used in a manufacturing system to produce two different product ranges. In the product range A (the net on the left), resource P_{c1} is used for activity P_{a2} and P_{c2} for activity P_{a3}. In the product range B (the net on the rigth), P_{c1} is used for P_{b3} and P_{c2} for P_{b2}. Place P_{a1} indicates that a new product A can be produced after activities P_{a2} and P_{a3} have been completed. Place P_{b1} has a similar meaning for the product range B. Both Petri net models in FIG. 3.30 are correct: they are bounded, live, and reversible.

FIG. 3.31 shows the asynchronous composition of these nets, obtained by merging places P_{c1} and P_{c2}. The resulting Petri net is bounded but neither live nor reversible. Sequences $s_a = t_{a1}t_{a2}t_{a3}$ and $s_b = t_{b1}t_{b2}t_{b3}$, which are fireable in the Petri net of each

product range in FIG. 3.30, are still fireable in the overall Petri net. However, after firing sequence $t_{a1}t_{b1}$ or $t_{b1}t_{a1}$ in the overall net shown in FIG. 3.31, leading to marking $P_{a2}P_{b2}$, a deadlock occurs, and no transition is enabled, highlighting a design error.

Let us analyze using the structural analysis. R_a has three conservatives components, $f_{a_1} = \{P_{a2}, P_{c1}\}$, $f_{a_2} = \{P_{a1}, P_{a2}, P_{a3}\}$, and $f_{a_3} = \{P_{a3}, P_{c2}\}$, and one repetitive component $s_a = \{t_{a1}, t_{a2}, t_{a3}\}$. R_b has three conservatives components, $f_{b_1} = \{P_{b3}, P_{c1}\}$, $f_{b_2} = \{P_{b1}, P_{b2}, P_{b3}\}$, and $f_{b_3} = \{P_{b2}, P_{c2}\}$, and one repetitive component $s_b = \{t_{b1}, t_{b2}, t_{b3}\}$. For the initial marking $M_a = P_{a1}P_{c1}P_{c2}$ in R_a, the sequence $s_a = t_{a1}t_{a2}t_{a3}$ is a realizable t-invariant, as is the sequence $s_b = t_{b1}t_{b2}t_{b3}$ for $M_b = P_{b1}P_{c1}P_{c2}$ in R_b.

The overall net R_o in FIG. 3.31 has $P_c = \{P_{c1}, P_{c2}\}$, $P_{ai} = \{P_{a1}, P_{a2}, P_{a3}\}$ and $P_{bi} = \{P_{b1}, P_{b2}, P_{b3}\}$. This net retains the preserved internal conservative components $\mathbf{f}_{a2}$ and $\mathbf{f}_{b2}$, and has two new ones, $f_{g_1} = \{P_{a3}, P_{b2}, P_{c2}\}$, and $f_{g_2} = \{P_{a2}, P_{b3}, P_{c1}\}$. The overall net is proved to be bounded. No new repetitive component emerges in the overall net, and it retains the repetitive components $s_a = \{t_{a1}, t_{a2}, t_{a3}\}$ and $s_b = \{t_{b1}, t_{b2}, t_{b3}\}$. These sets can be fired from the initial marking $P_{a1}P_{c1}P_{c2}P_{b1}$ and are therefore realizable t-invariants. However, the overall Petri net is not live: the firing of sequence $t_{a1}t_{b1}$ (or $t_{b1}t_{a1}$) leads to a deadlock.

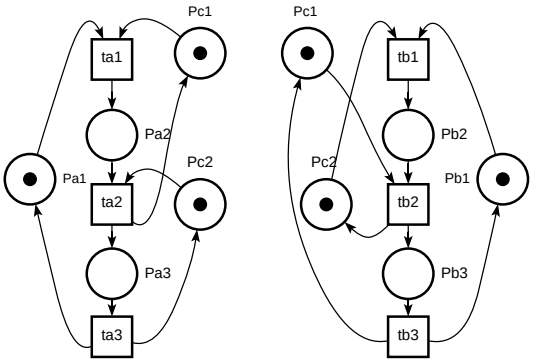


Figure 3.30: Two Petri nets.

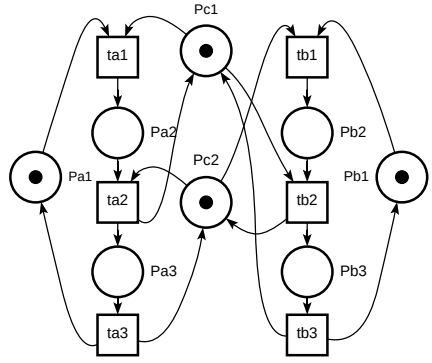


Figure 3.31: Overall Petri net.

In this example, an error may have occurred in the modeling of the product range B. If P_{c1} is actually used for P_{b2} and P_{c2} for P_{b3}, the overall Petri net is as shown in FIG. 3.32. In this case, no deadlock occurs, and the overall Petri net is live, as well as bounded and reversible. This net retains the preserved conservative components $\mathbf{f}_{a2}$ and $\mathbf{f}_{b2}$, and introduces two new ones, $f_{g_1} = \{P_{a3}, P_{b3}, P_{c2}\}$, and $f_{g_2} = \{P_{a2}, P_{b2}, P_{c1}\}$. It also retains the repetitive components of R_a and R_b, which correspond to the firing sequences $s_1 = t_{a1}t_{a2}t_{a3}$ and $s_2 = t_{b1}t_{b2}t_{b3}$ from the initial marking $M = P_{a1}P_{c1}P_{c2}P_{b1}$.

◇

A more complex example is provided in Section D.2 demonstrating how the intersection of sections of the transportation system from Example 1.1 is derived through the asynchronous composition of sections.

3.7.3 Communication by token passing

A particular case of asynchronous composition is communication by token passing. A message sent by a Petri net corresponds to an output place, and a message received

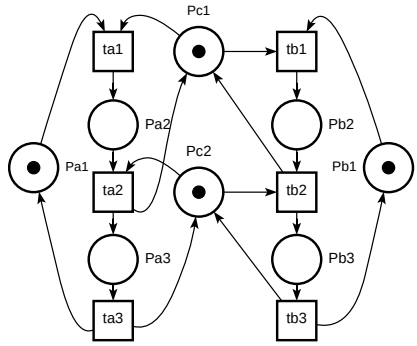


Figure 3.32: Asynchronous composition with a different design.

corresponds to an input place. These two places are merged during asynchronous composition. This approach can be used to build an overall system made up of elements which communicate by means of messages passing through buffers (the merged places).

Example 3.22 Token passing
Consider two machines, A and B, that must communicate with each other to execute its own operations, as represented on FIG. 3.33:
- A can start only after B has finished;
- B can start only after A has already started.

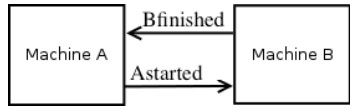


Figure 3.33: Two communication machines.

If the two machines A and B were independent, the Petri net model would be the one represented on FIG. 3.34. However, this is not the case; they need to communicate, and communicating places are added to each model, as depicted on FIG. 3.35. In the machine A model, place $p3$ is associated with predicate *B has ended*, and $p4$ is associated with predicate *A has started*. In the machine B model, place $q4$ is associated with predicate *B has ended*, and $q3$ is associated with predicate *A has started*. The overall model on FIG. 3.36 is obtained by merging places having the same predicate: $p3$ and $q4$ into place $p4q3$, as well as places $p4$ and $q3$ into place $p4q3$.

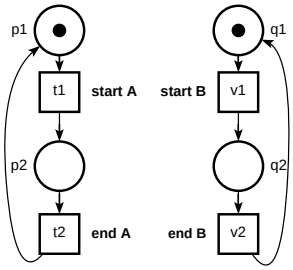


Figure 3.34: Petri nets.

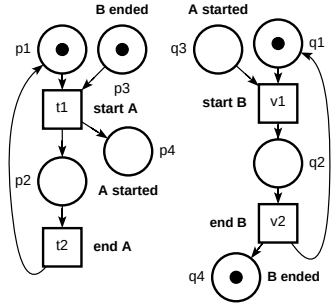


Figure 3.35: Adding places.

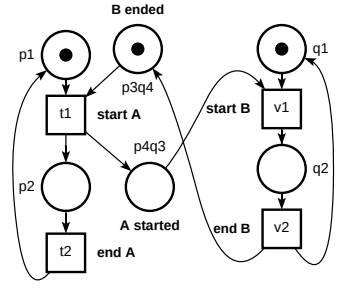


Figure 3.36: overall PN.

The composition does not guarantee that the overall obtained model will retain the properties of the initial modules. It is necessary to analyse the overall model. In the case of this example, and for this marking, the overall model retains the three properties. ◇

Notice that in this particular case of asynchronous composition, only the initial and final models, shown in FIG. 3.34 and FIG. 3.36, need to be analyzed. The model shown in FIG. 3.35 will always be non-live: t_1 can fire once from the initial marking $M = p_1 p_3$.

3.8 Notes

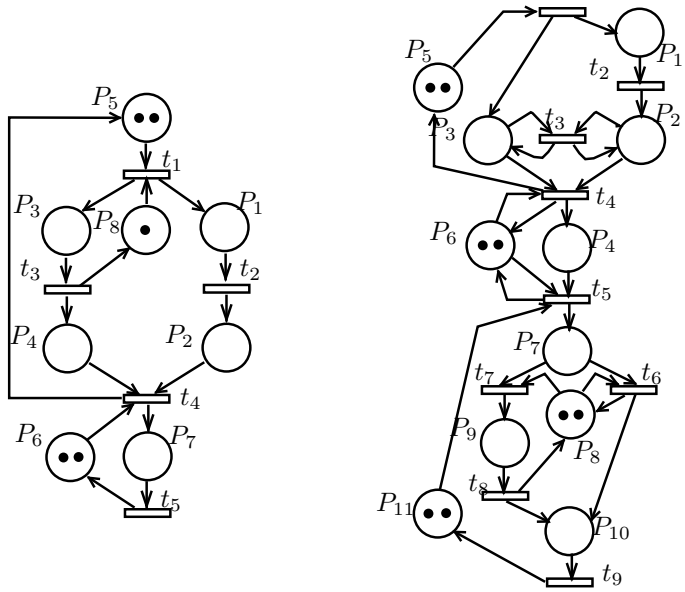
More information about Petri net reduction can be found in BERTHELOT (1987).

If you used the Tina toolbox to assist you with Petri net editing and simulation in the exercises in Chapters 1 and 2, you can use it now for the analysis by marking enumeration as well as structural analysis. Refer to Appendix E for a short manual.

3.9 Exercises

1. Based on the results obtained in Exercise 1 of chapter 2, analyze the behavior of each Petri net. Interpret the results obtained, especially the invariants.

2. Analyze the Petri nets in FIG. 3.37 using the reduction method. Detail each step, naming the rule and justifying its application.



a) b)

Figure 3.37: Analysis by reduction.

3. Consider the Petri net in FIG. 2.30.a with the initial marking $M_0 = P_1P_4$ and FIG. 2.30.b with the initial marking $M_0 = P_1P_6$ and their respective conservative and repetitive components, calculated in Exercise 4 proposed in chapter 2.
   a) What can be deduced about the properties of the Petri net? Is the net conservative? Is it repetitive?
   b) For the initial marking M_0 and the already calculated invariants, what additional information is there about the properties? Is it possible to determine if the Petri net is live and bounded?
   c) Find the coverability tree of this net. If there is a deadlock, provide the sequence that produces it, starting from the initial marking;
   d) For the Petri net in FIG. 2.30.b, find the sequence that allows firing t_3 and compare it with question a).

4. Consider the Petri net model in FIG. 3.38, which represents one section of the transportation system discussed in Example 1.1. Transition t_2 represents the moment when the vehicle reaches the contact, which causes it to stop (i.e., it is no longer moving). The Petri net model in FIG. 3.39 introduces the notion of a cell, which is occupied only when the vehicle is in motion within the section. Identify which transitions correspond to the same events, and create a synchronous composition of these models.

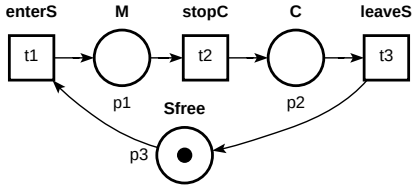


Figure 3.38: Petri net of a section.

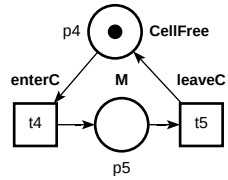


Figure 3.39: Petri net of a cell.

Part II

Data, Time, and External Environment

Chapter 4

Interpreted Nets: data and time

The first part of this book defined the basic model of Petri nets, also called ordinary Petri nets. Only the structure is represented, and conflicts, for example, are not resolved. It models only the causal relationship between actions: the event *start of operation* happens before the event *end of operation*. In binary Petri nets, the token only indicates whether the proposition associated with the place is true or not. Generally, the token acts as a counter.

Petri nets can be used to describe *sequences* of processes acting on complex data structures, such as signals that allow communication with the external environment (like sensors and actuators) and explicit temporal constraints. This data structure can also be a hierarchy of object classes (for example, the *part* class, with attributes: operation order, machines where it will be processed, etc.). In this case, it is a high-level Petri net, the subject of Chapter 5.

In the case of interpreted Petri nets, variables are associated with the transitions of the net — representing conditions and actions present in the system. These variables can indicate the state of actuators, sensors, etc., thus allowing the modeling of interaction with the external environment.

In an ordinary Petri net, as in FIG. 4.1, a transition may be fired when it is enabled. Enabling is a necessary and sufficient condition for firing: if there is no conflict, the transition fires. When there is a conflit, the non-determinism can be resolved, for exemple, by associating a priority with the transitions, as presented in Section 2.2. Then the chosen transition is fired.

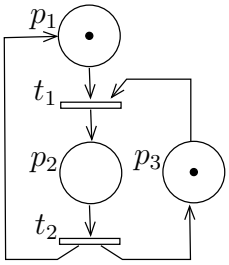


Figure 4.1: Ordinary PN.

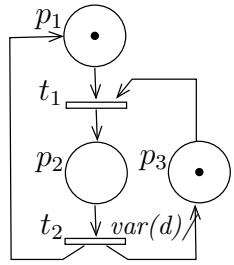


Figure 4.2: With time.

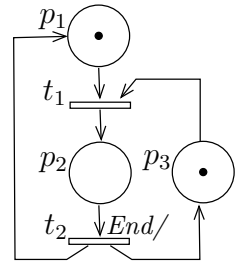


Figure 4.3: With data.

The Petri net in FIG. 4.1 is live, bounded, and reversible. Transition t_1 is enabled for the initial marking $M = p_1p_3$ and fires (the firing is indivisible and has zero duration), leading to a new marking $M' = p_2$. In turn, t_2 is enabled and fires, returning the net to

the initial marking.

Suppose place p_1 represents *parts waiting*; place p_2, *part being worked on*; place p_3, *machine free*; transition t_1 represents *start of operation* and t_2, *end of operation*. As long as places p_1 and p_3 are marked, the events *start* and *end* of operation, corresponding to the sequence t_1t_2, will occur.

In the real system, the time between the start of the operation (firing of t_1), when place p_2 is marked, and the end of the operation (firing of t_2), corresponds to the duration of the operation and is therefore non-zero.

There are different ways to represent the occurrence of events associated with the start and end of the operation, whether for simulation, emulation, or implementation purposes.

- Elapsed time: To correctly represent the behavior of a system where the elapsed time is important, it is necessary for the model to consider temporal information. Time can be considered explicitly through a Boolean variable whose value depends on time. Consider FIG. 4.2 and let d be the duration of activity *part being worked on* (place p_2). If $var(d)$ is associated with transition t_2, this transition will fire only if, in addition to place p_2 being marked, the time d has elapsed after t_1 firing. Note there is no information coming from the system.

- Information from the external environment: A sensor in the system (real or simulated) can be represented by a Boolean variable associated to a transition, indicating the state of this sensor, and used as an additional firing condition. Consider FIG. 4.3 and let *End* be a Boolean variable indicating the state of an *end of operation* sensor. Transition t_2 will fire only if, in addition to place p_2 being marked, $End = 1$.

In both cases, $M(p_2) = 1$ is a necessary condition for the firing of t_2, but it is not sufficient.

4.1 What is Interpretation?

4.1.1 Semantics of Places, Transitions, and Tokens

Interpreting a Petri net means, above all, giving a concrete meaning to a mathematical model by associating the places, transitions, and tokens with elements existing in the system.

As seen in Chapter 1, places can be interpreted as activities in a discrete event system, part stocks and resource states in a manufacturing system, logical conditions in a sequential logic system, tasks being executed in a computer system, etc.

A transition can correspond to an event considered to have zero duration, as indicated in FIG. 4.4.a. It can also be associated with an activity or task, provided that it is *indivisible* or *non-interruptible*. Such a transition is, in this case, called an *abbreviation* of an elementary sequence, consisting of a start transition t_1, a place p describing the activity or task, and an end transition t_2, as indicated in FIG. 4.4.b. The two Petri nets in this figure have the same properties (boundedness, liveness, reversibility) because place p is a *substitutable* place (see reduction rules in Chapter 3).

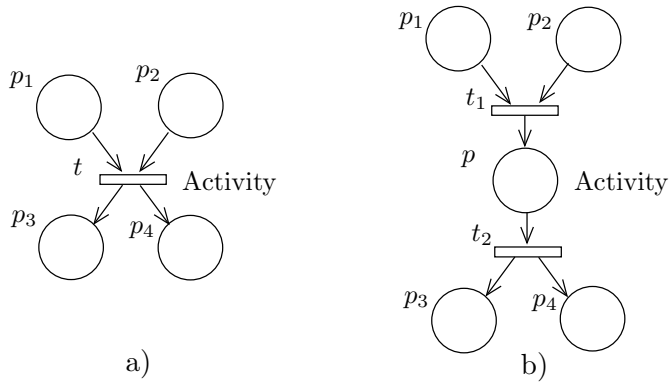


Figure 4.4: Association of an activity with a transition.

Tokens can be interpreted as physical objects, information, data structures, or resources. Such entities are subjected to events (firing of transitions) that cause them to change state (move from one place to another).

4.1.2 Interaction with data and the environment

Modeling an *open* system (a system constantly communicating with the external environment) using Petri nets involves structuring it into two parts: control and data.

The *control structure* of the modeled system is defined by the structure of the Petri net describing the sequences of processes to be performed by a system.

These processes must be specified during modeling, complementing the information contained in the Petri net structure. This requires associating the specification of processes with transitions or places. In the former case, the process executes uninterruptedly when the transition fires; in the latter, it unfolds as long as the place is marked.

Sometimes, certain processes are not systematically executed immediately after the completion of the preceding processes. Their execution may be subject to conditions that deal with certain data such as conditional instructions (e.g.,*"if-then-else"*). Such a behavior is described by a place with multiple output transitions. From the perspective of Petri net theory, this characterizes a situation of *conflict*. The firing of one of the conflicting transitions occurs when some *supplementary firing condition* is true.

In the case of ordinary Petri nets, this supplementary firing condition can be a priority, as seen in Section 2.2. In the case of systems that interact with their environment, supplementary firing conditions depend on data.

The *data structure* (also known as the *operational structure*) simultaneously describes the internal data structures of the system and the computations performed on these data, without specifying when they will occur. The data can be categorized into: internal data, temporal information, or external data, as sensors, actuators and message reception/transmission.

The interpretation of the Petri net specifies the connections between the *control* structure and the *data* structure, as illustrated in FIG. 4.5. The interpretation includes the association of conditions and action to transitions, and treatment to places.

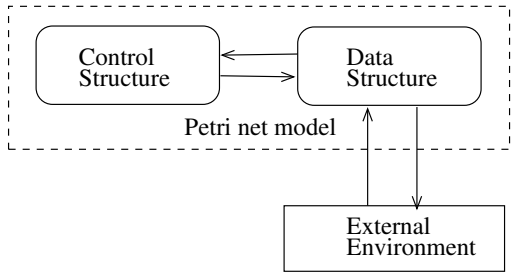


Figure 4.5: Interaction of the Petri net with the external environment.

4.1.3 Interpreted Petri Net

An interpreted Petri net (IPN) is a triple

$$IPN =< R, D, I > \tag{4.1}$$

where:

- R is a Petri net as defined in Section 2.1.1, also called Petri net structure or underlying Petri net;
- D is a data structure, representing internal and/or external data;
- $I : T \rightarrow D$ is the interpretation that associates with each transition $t_i, i = 1..\ |T|$ a pair (C_i, A_i) of conditions C_i and actions A_i on the data structure D.

Both conditions C_i and actions A_i can be expressed as conjunctions of Boolean variables. Graphically, they are represented in the form (C_i, A_i) or C_i/A_i next to transition t_i.

The interpreted Petri net provides a good structuring of the *control*. However, there is no method for structuring the data. Chapter 5 presents Petri net models that propose data structuring: predicate-transition Petri net, colored Petri net, and object Petri net. Such Petri nets allow a compact and structured description, based on the fact that tokens are individual objets with data attached to them.

4.1.3.1 Fireable transition

A transition t is fireable in an interpreted Petri net if and only if:

- Transition t is enabled in the underlying Petri net as defined by Equation2.6,
- Condition C associated with t is true.

4.1.3.2 Transition Firing in an Interpreted Petri Net

If a transition t is fireable, its firing:

- Leads to a new marking M' in the underlying Petri net using Equation 2.9,
- Executes the action A associated with t.

4.1.4 Explicit Consideration of Time

The explicit consideration of time is achieved through Boolean variables appearing in both conditions and actions associated with transitions and whose values depend on time.

Let $var(\theta)$ be a Boolean variable; assume its initial value is "0" and that it automatically takes the value "1" after a duration θ. Suppose that when transition t_1 is fired in FIG. 4.2, such a variable takes the value "0". If a transition t_2, immediately enabled by the firing of t_1, has the condition

$$var(\theta) = 1,$$

then the time interval between the firing of t_1 and t_2 will be exactly θ. This interval corresponds to the duration of the activity associated with place p_2.

Time can also be considered explicitly in the model itself, in a rigorous and systematic manner, by associating a duration with transitions (or places). Some of these Petri net models — timed, temporal, and stochastic — will be presented in Chapter 6.

4.2 Analysis

The underlying Petri net only describes the *control structure* of the system. Thus, the marking of the Petri net only provides the *control state*. The *system state*, that is, the state of the interpreted Petri net, is described by the marking associated with the state of the *data* at a given time.

interpreted Petri net state	=	marking + data state + time elapsed

It is necessary to observe that the evolutions of the markings of the interpreted Petri net are restrictions of the evolutions of the markings of the underlying net (without its interpretation). Indeed, data may prevent the firing of an enabled transition but not allows the firing of a transition which is not enabled. This means that the set of reachable markings with interpretation is included in the set of reachable markings without interpretation. This is illustrated in FIG. 4.6.

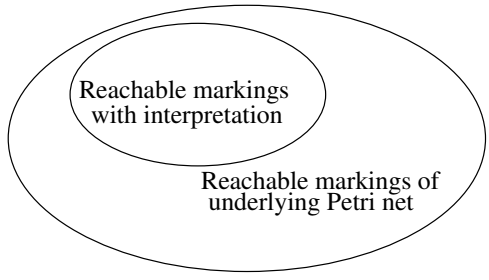


Figure 4.6: Reachable markings of an IPN and its underlying net.

Therefore, we have the following results:

underlying Petri Net		Interpreted Petri Net
k-bounded for M	$\Longrightarrow$	k-bounded for M + data
f place invariant	$\Longrightarrow$	f place invariant

On the other hand:

underlying Petri Net		Interpreted Petri Net
live	$\neq$	live
reversible	$\neq$	reversible

One may then question the interest of analyzing the underlying Petri net of an IPN, considering that its properties may change.

The first reason is that the properties of interpreted Petri nets are undecidable. Indeed, the complexity of verifying these properties is the same as formal program verification. However, certain properties (in particular, place invariants) are preserved, which facilitates subsequent reasoning regarding the interpreted net.

The second reason is that the design and validation of a complex system must be decomposed into stages. Studying the underlying Petri net, which is easier, allows inconsistencies to be identified early on and avoids the high cost caused by errors in the specification phase. It would not make sense to build a data specification based on an underlying net with an inconsistent structure, such as a deadlock, for example.

4.3 Validation by Simulation

Therefore, the first step is to analyze the underlying net of the interpreted Petri net as seen in Section 4.2. To obtain more information about the dynamic behavior of the interpreted net, it needs to be simulated. Although simulation is a non-exhaustive validation method (as it is impossible to predict *all* possible configurations of variables), it allows testing the behavior of the modeled system for the envisaged configurations.

The simplest way to simulate a control system is for the user to provide step-by-step values for the variables associated with transitions. If the simulation is performed using the actual response time of each system, it becomes an *emulation*. In this validation phase, the system is first simulated and then emulated because the behavior may change when using actual response times.

For discrete event systems, particularly systems that interacts with its environment, it is essential for the designer to have an integrated environment of specification, validation (through analysis and simulation), and implementation tools. The Petri net, being a formal description technique, allows for this integration.

4.4 Modeling with Interpreted Petri Nets

The steps to model a system using interpreted Petri nets are as follows:

1. Determine the structure of the Petri net (i.e. underlying net), representing concurrent, parallel, sequential activities, etc. Describe at the same time which events are associated with the transitions and how to represent them, particularly indicating

the events that require interaction with the external environment. These pieces of information are not part of the net structure but will be part of the data of the interpreted Petri net.

2. Analyze the underlying net, checking if it has good properties, such as the absence of deadlock. If necessary, correct both the structural and data parts.

3. Simulate the interpreted net (control and data) to try to obtain the behavior of the modeled system. Remember that a simulation provides results only about the chosen set of input values (initial marking and initial values of the variables) and that, in almost all cases, it is not possible to simulate all possible input configurations, either due to lack of time or forgetfulness. Therefore, it is impossible to guarantee that the system will always behave in the same way.

The boundaries between what should be modeled in the control part and in the data part are imprecise and delicate choices are to be made during the design.

Progressively, one moves from an *abstract model* to a detailed description, refining the description through the enrichment of the interpretation.

4.5 Example

To illustrate the modeling of a system using interpreted Petri nets, we will present the example discussed in SOARES (1994), concerning the automation of an oil collection station similar to those existing in the land fields explored by Petrobrás in Rio Grande do Norte.

The function of a collection station is to receive crude oil from wells located in its vicinity, test the flow rate of each of these wells, and transfer the produced volume through transfer pumps via pipelines leading to larger capacity storage stations.

4.5.1 Process Description

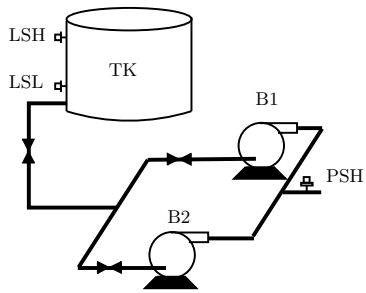


Figure 4.7: Oil collection station.

The system depicted in FIG. 4.7 features a storage tank (TK) that continuously receives oil through a pipeline connected to the producing wells. The tank has two level switches (sensors) LSH and LSL for indicating that the oil level is, respectively, high and

low. There are two pumps B_1 and B_2 connected to a pressure switch (sensor) PSH that signals high pressure at the pump discharge.

When the oil reaches the high level in the tank (LSH=1), a motor pump is activated to transfer the oil to a central collection station. A standby motor pump should be activated if the motor pump is not turned on within a 2-second interval. The motor pump (B_1 or B_2) should be turned off in two cases:

- When the oil level in the tank drops to the low level (LSL=1) to prevent it from running dry (which could damage it).
- When there is an excessive pressure increase in the pump discharge (PSH=1), which indicates a blockage in the downstream pipe.

To automate this system, the following tasks are necessary:[1]

- Interlocking of the level switches and pressure switch with the pumps.
- Measurement of the produced volume through counting electrical pulses sent by a flow meter (not represented in FIG. 4.7).
- Acquisition and transmission of data to a central station, where supervisory control of multiple stations is performed.

Interlocking Logic

Pumps B_1 and B_2 are alternately activated for each transfer. If B_1 is activated for the first transfer of the day, B_2 will be activated for the second transfer, and so on (indicated by the variable `nxt`). To activate a selected pump B_i, it must be operational (B$i$ok), and the tank must have reached the high level (LSH=1). The shutdown of the activated pump is triggered when the tank reaches the low level (LSL=1), indicating the end of the transfer.

When a specific pump B$_i$ is activated with the action turnOnBi, the variable `nxt` is updated to reflect this new activation. The program then waits for 2 seconds for the confirmation signal B$_i$IsOn (from an auxiliary contact of its contactor) indicating that the pump is running. If, after 2 seconds (bi(2)=1), the program does not receive the confirmation (B$_i$IsOn=1), the pump is turned off (turnOffBi), and it is marked as no longer operational (B$i$ok=0).

The high pressure signal (PSH) sent by the pressure switch in the pump discharge during transfer causes the shutdown of the pump in operation. The current pump must be turned off (turnOffBi) and considered as no longer operational (B$i$ok=0). This signal should be ignored if it occurs within 30 seconds of pump activation to prevent transient high pressure from unnecessarily interrupting the transfer.

[1] In practice, these tasks are implemented through programmable logic controllers (PLCs) located at each collection station. Although the programming language of PLCs is familiar and easy to understand for the technical staff involved, it does not allow validation of the application program, which is typically done through testing with the real system.

Sensors	
LSL	oil level in the tank is low
LSH	oil level in the tank is high
PSH	pump pressure is high
BiIsOn	pump B_i is switched on
Actuators	
turnOnBi	switch on pump B_i
turnOffBi	switch off pump B_i
Internal variables	
nxt	next pump to be used
B$i$ok	pump Bi is operational (there is no failure)
bi(θ)	θ sec was elapsed since Bi was switched on
psh(θ)	θ sec was elapsed since PSH was received

Table 4.1: Data associated with the Petri net.

4.5.2 Interpreted Petri Net Model

FIG. 4.8 represents the interpreted Petri net of the control system model. The data structure is given in Table 4.1. Transitions are associated with a pair C_i/A_i as defined in Section 4.1.3. The conjunction of actions or conditions is represented by a comma instead of the mathematical symbol $\wedge$. Some transitions, such as t_4, do not have associated actions. Others, like t_{11}, do not have conditions and may fire as soon as they are enabled by the marking.

The initial marking in FIG. 4.8 is $M_0 = p_1$: both pumps are off and the controller is waiting for the reception of the sensor value LSH=1, indicating that the oil in the tank reached the high level. To validate this model, the first step is to analyze the underlying Petri net. The underlying Petri net of FIG. 4.8 is live, reversible, and binary.

The initial state of the data is as follows: pump B_1 was chosen to be activated in the first transfer of the day (`nxt`=B1), both pumps are operational (B_1ok=1, B_2ok=1), and the oil level in the tank is low (LSL=1). All other variables in Table 4.1 have a value of 0 (or False).

For an efficient validation of an Interpreted Petri net, it is necessary to simulate both the control system and the oil collection station. The control can be simulated by a token player[2] with a data structure, as represented in FIG. 4.5. The plant needs to be simulated by a hybrid simulator that handles the continuous part (oil level fluctuations) as well as the discrete part (pressure switch).

Let us consider some scenarios from the initial state described above:

- Scenario 1 (No failures): After a while, the oil reaches the high level, pump B1 is turned on, and at $\theta = 1.5$ seconds, the confirmation signal is received for pump B1. Later, the low oil level is reached, and B1 is turned off. This scenario corresponds to the following firing sequence from the initial marking M_0 described above: $t_3t_5t_7t_{11}$.

- Scenario 2 (Failure of pump B1): When the oil reaches the high level, pump B1 is turned on, but it is not activated within 2 seconds. Therefore, pump B2 must be turned on, which operates without failure. Pump B1 is later repaired by a human operator. This corresponds to the following firing sequence from M_0: $t_3t_2t_8t_4t_6t_{12}$.

[2]See Chapter 7 and FIG. 7.1.

- Scenario 3 (Failure of both pumps): When the oil reaches the high level, pump B1 is turned on but not activated within 2 seconds. Pump B2 is then turned on but also fails to activate within 2 seconds. Pump B1 has not yet been repaired by a human operator. The corresponding firing sequence is: $t_3t_2t_8t_1$. With no pump capable of transferring the oil from the tank that has reached the high level, this situation presents a serious problem if at least one of the pumps is not repaired.

When validating a model, it is important to simulate critical special cases like the one depicted in Scenario 3. Other interesting scenarios are proposed in Exercise 1.

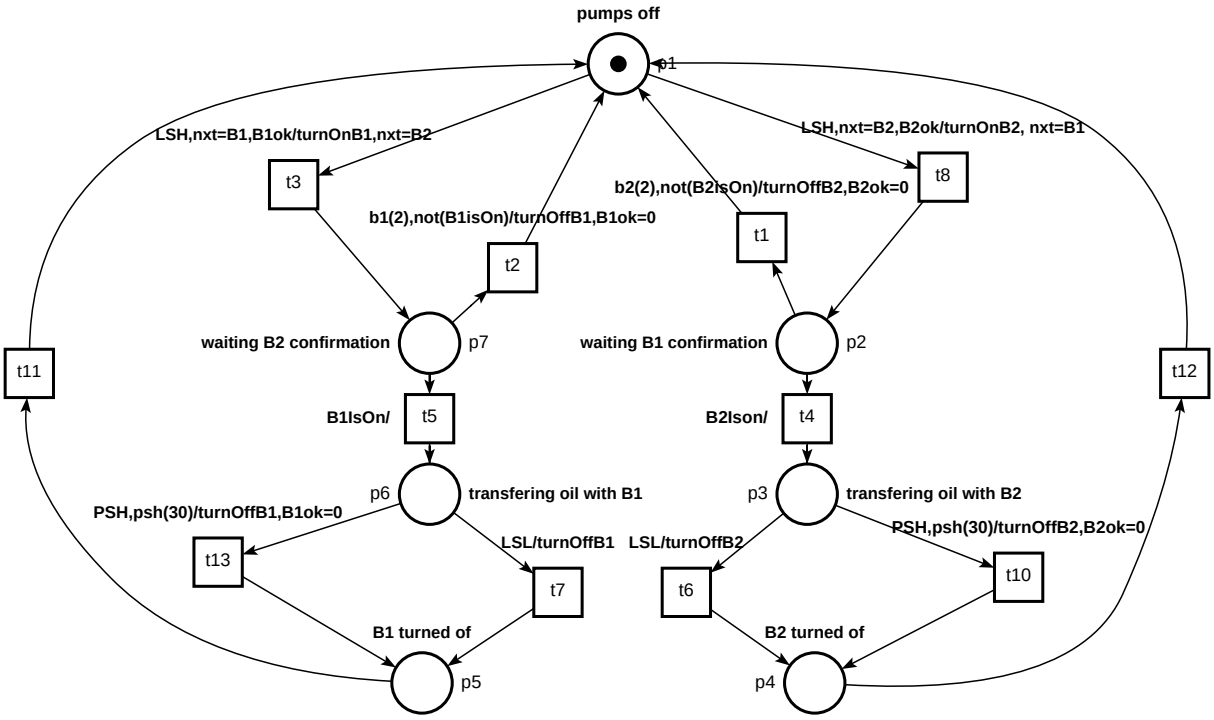


Figure 4.8: Interpreted Petri net of the control.

It's worth noting in FIG. 4.8 that the structure of pump B1 control is the same as that for pump B2 control. If additional pumps are added, it would suffice to add one structure per pump and modify the conditions and actions of the new transitions. However, in a system with a large number of pumps, the Petri net could become quite large.

4.6 Exercises

1 Propose other simulation scenarios for the example presented in Section 4.5 such as:

1. Pumps are correctly activated, but there there is a transient high pressure in the pump discharge: $PSH = 1$, but $psh(30)$ is always 0 at time $\theta = 30$.
2. Pumps are correctly activated, there is a high pressure in the pump discharge, $PSH = 1$, and $psh(30) = 1$ at time $\theta = 30$. It was not a transient failure and the pump must be turned off.

2 Model the exercises proposed in Chapter 1 using the interpreted Petri net model.

Chapter 5

High-Level Petri Nets

The following models propose a structuring of the data part of the system, distinguishing, for example, global data from local data. Local data are understood to be data that only intervene in certain conditions or actions and are only accessible at certain times. Colored nets, predicate-transition nets, and object nets structure part of the data with the tokens.

This chapter will present the main elements underlying these models: the concept of folding of places and transitions, and that of the token as a carrier of information. Simplified definitions will be given to differentiate the models from each other and from the ordinary net. Formal definitions can be found in the references at the end of the chapter.

The boundary between control and data in a system is not inherent to the described system. It results from a designer's choice. At one extreme, almost all control can be integrated into the execution conditions of the treatments, thus disappearing from the Petri net structure. At the other extreme, the control can be overly developed, resulting in large Petri nets that are difficult to handle.

Complex systems, such as manufacturing systems, are characterized by the ability to produce various types of products simultaneously. Therefore, there is a significant data component (type of part to be manufactured, current state of the part, etc.), in which dynamics play a fundamental role (sequence of possible operations on a manufacturing process, for example). The allocation of resources (machines, transportation system) and the complexity of allocation mechanisms highlight the parallelism (both cooperation and competition) existing in the system. Due to these characteristics, the Petri net is considered a suitable tool for specifying the control of a manufacturing system. Moreover, its formalism allows for validation through analysis, as seen in chapter 3, and simulation.

However, in such complex systems, some problems arise. This complexity sometimes means the composition of several similar processes. In this case, when using the ordinary Petri net (with the marking of places given by undifferentiated tokens, and with places behaving like counters), there are two choices:

- Model the general behavior without specifying the identity of each process, but only their number;
- Model each process that constitutes the system individually and model the interaction between them, which often involves unfolding the model that represents the general behavior.

In the first case, a compact description is obtained, but it is not detailed enough: there is a lack of information. In the second case, the obtained model may be impractical to work with, either because of the size of the net or the number of existing interactions. Therefore, it is necessary to structure part of the system's data outside the net structure.

Several models have been proposed, referred to in this book as High-Level Petri Nets (HLPN). These models, although more or less equivalent, each have their own characteristics. The next section will present the general characteristics of HLPN, and then the main characteristics of each model. Finally, the models will be compared, highlighting the common points.

5.1 General Characteristics

To better discuss the characteristics of an HLPN, an example of coordination of parts passing through different machines in a manufacturing system will be presented.

Example 5.1 Three similar machines
The system has three machines M_1, M_2, M_3. The parts that enter the workshop at a given time can belong to different manufacturing processes, each process being characterized by a sequence of operations. If the machine to which the part was assigned is free, the operation begins. After its completion, the machine is freed, and the part is ready for the next operation. To simplify the modeling, it is assumed that the entry/exit of parts from the workshop is done by a loading/unloading machine.

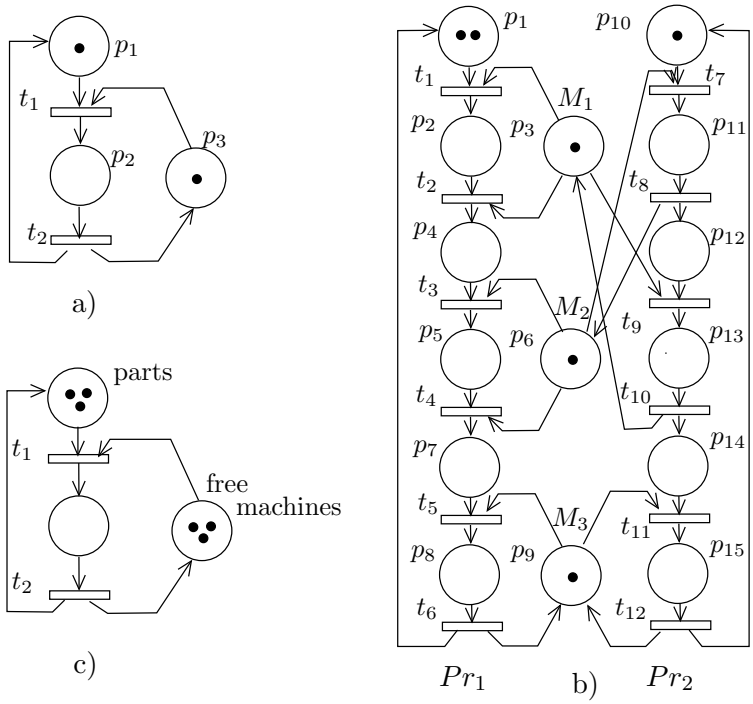


Figure 5.1: Behavior: a) One machine. b) Three machines. c) General.

The FIG. 5.1.a shows the (ordinary) Petri net in the case of a single machine and a single part. FIG. 5.1.b represents the more complex case of three parts belonging

to two different manufacturing processes, Pr_1 and Pr_2. Each process consists of three operations, performed by different machines (M_1, M_2, and M_3). The sequence of parts through the machines is M_1–M_2–M_3 for process Pr_1 and M_2–M_1–M_3 for process Pr_2.

This is a typical case of a system where the processes have similar behavior. To simplify the model of FIG. 5.1.b, one can represent only the general behavior of the system, the number of processes being given by the number of tokens in the place *free machines* (FIG. 5.1.c). However, this model does not indicate which machine is manufacturing which part, nor does it show which machine is available or which part is waiting. If there is one token in the *parts waiting* place and one token in the *free machines* place, transition t_1 (*start operation*) can fire without being able to verify which machine is assigned to the piece. ◇

This shift from a detailed model, showing all processes, to a model representing only the general behavior is called *folding* the net. In fact, the set of processes with the same structure (or similar structures) is *folded* into a single conservative component.

Example 5.2 Readers and writer (folded)
A well-known example of a system with multiple processes exhibiting identical behavior is the readers and writers, modeled by the Petri net in FIG. 5.2. The challenge is to manage a shared resource, such as a data structure, noted by S in the Petri net model. A similar model was presented in Example 2.1.

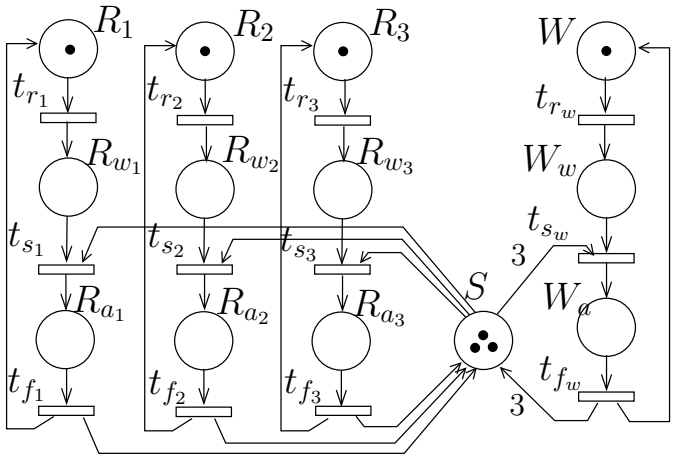


Figure 5.2: Readers and writers: detailed Petri net.

The places $R_i, i = 1..3$, indicate that the read process is inactive. When there is a read request (firing of transition t_{r_i}), the process is waiting (places R_{w_i}). If the resource is free at this moment, one of the processes can start reading (firing one of the start reading transitions, t_{s_i}), moving to the active state (place R_{a_i}). Once the reading is finished, the corresponding transition t_{f_i} fires, and the system returns to the original marking. This system can read up to three processes simultaneously, but it enforces mutual exclusion between read and write processes. The transitions and places for the write process are analogous to those for the read processes; the arcs (S, t_{s_w}) and (S, t_{f_w}) have a weight of 3 to prevent a reader from reading a memory zone at the same time it is being written to.

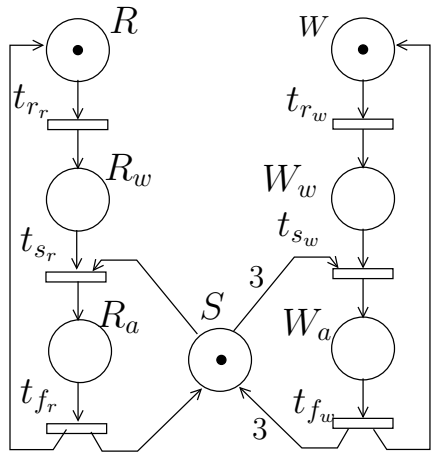


Figure 5.3: Readers and writers: folded Petri nets.

Since the read processes are similar, they can be represented by a single process with three tokens (one for each process), as shown in FIG. 5.3. The places R_i are folded into place R, the places R_{w_i} are folded into place R_w, and the places R_{a_i} are folded into place R_a. The transitions t_{r_i} are folded into t_{r_r}, the transitions t_{s_i} into t_{s_r}, and the transitions t_{f_i} into t_{f_r}. The net becomes simpler since the three *reader* processes are described using the same structure (a sequence of places and transitions). However, when an event involving a reader occurs, the transition t_{r_r} fires, but it is not clear which reader process is involved. ◇

In this way, a more compact and abstract model is obtained, but some information is lost. Simply *folding* a net may not be sufficient and can even misrepresent the system. Therefore, simply folding the net is insufficient. In addition to making the model more compact, it is particularly important to distinguish or individualize each token, enabling it to carry specific information.

The underlying idea of Petri nets with token individualization is precisely to achieve this folding without losing information and without losing the graphical representation of the process structure. Thus, with the concepts of folding a net and token individualization, the expressive power of the Petri net can be increased.

To these two characteristics, one more is added: the use of variables or functions in the firing of transitions. Now that each token is an individual, it is necessary to choose which ones will enable the transition and make the system evolve.

The following sections present the different Petri net models. They all rely on the idea of net folding with token individualization. Depending on the form this individualization takes, there is the colored Petri net model, the predicate-transition net, and the object Petri net.

5.2 The Different HLPN Models

5.2.1 Colored Petri Net

5.2.1.1 Formal (simplified) definition

A colored Petri net associated with an initial marking is a sextuple given by:

$$N_c =< P, T, C_{col}, C_{sc}, W, M_0 >$$

where:

- P is a finite set of places;
- T is a finite set of transitions;
- C_{col} is a finite set of colors;
- C_{sc} is the subset of colors function that associates a subset of C_{col} with each place and transition: $C_{sc} : P \cup T \longrightarrow \mathcal{P}(C_{col})$;
- W is the incidence function (equivalent to $C = Post - Pre$). Each element $W(p,t)$ of W is also a function that associates each place-transition pair, $W(p,t) : C_{sc}(t) \times C_{sc}(p) \longrightarrow I\!N$;
- M_0 is the initial marking that associates, for each place and for each possible color in this place, a number of tokens: $M_0(p) : C_{sc}(p) \longrightarrow I\!N$.

5.2.1.2 Association of Colors to Tokens

To differentiate tokens, colors (integers or sets of labels) are associated with them. Consequently, each *place* is associated with the set of colors of the tokens that can belong to that place. Each *transition* is associated with a set of colors that correspond to the different ways of firing a transition. In simpler cases, when all processes have exactly the same structure and are independent of one another, the colors of the transitions are directly associated with the processes, and the set of colors for places and transitions are identical.

For example, in the case of readers and writers in FIG. 5.3, the processes correspond to the identities of the readers. Therefore, the places R, R_w, and R_a and the transitions t_r, t_s, and t_f are associated with the set of colors R_1, R_2, R_3. This is not the case when places are shared by multiple processes. It is then necessary to introduce *composite colors*.

The set of colors of a transition indicates the different ways it can fire. Each transition color corresponds, in this case, to one of the transitions of the equivalent ordinary Petri net. In other words, each transition of the ordinary Petri net that is folded into a transition of the equivalent colored Petri net will correspond to a color in the set of colors of the latter.

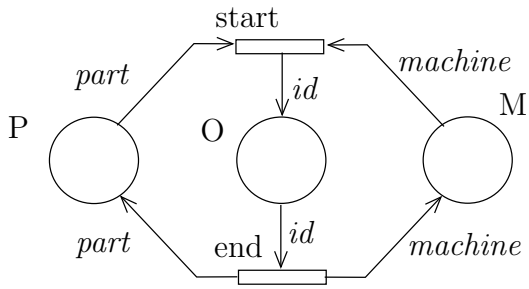


Figure 5.4: Colored Petri net.

Example 5.3 Colored Petri net
Consider again the manufacturing system example from Example 5.1, consisting of three machines and three parts. FIG. 5.4 represents the colored Petri net model of this system.

The set of colors for the place P is the set p_1, p_2, p_3, representing the names of the parts, and the set of colors for the place M is the set of machine names m_1, m_2, m_3. The colors for the place O form the set $p_1.m_1, p_2.m_1, p_3.m_1, p_1.m_2, \cdots, p_3.m_3$, which corresponds to the different machining operations and will be denoted in a simplified way as $o_{11}, o_{21}, o_{31}, o_{12}, \cdots, o_{33}$. The colors of the transitions *start* and *end* are the same as those of O.

The complete model of the colored net in FIG.5.4 is given by:

- $C_{col} = \{p_1,\ p_2,\ p_3,\ m_1,\ m_2,\ m_3,\ o_{11},\ o_{12},\ o_{13}, \cdots,\ o_{32},\ o_{33}\}$;
- $C_{sc}(P) = \{p_1,\ p_2,\ p_3\}$,
  $C_{sc}(M) = \{m_1,\ m_2,\ m_3\}$,
  $C_{sc}(O) = C_{sc}(start) = C_{sc}(end) = \{o_{11},\ o_{12}, \cdots,\ o_{32},\ o_{33}\}$;
- $W = \begin{array}{c} \\ P \\ O \\ M \end{array} \begin{array}{c} \begin{array}{cc} start & \quad end \end{array} \\ \begin{bmatrix} -part & part \\ id & -id \\ -machine & machine \end{bmatrix} \end{array}$

  whose functions are given by: $\begin{cases} id(o_{ij}) = o_{ij} \\ machine(o_{ij}) = m_j; \\ part(o_{ij}) = p_i; \end{cases}$

- $M_0 = \begin{array}{c} \begin{array}{cccccccccccc} p_1 & p_2 & p_3 & m_1 & m_2 & m_3 & o_{11} & o_{12} & o_{13} & o_{21} & \cdots & o_{33} \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \end{array},$

  or, in the form of a vector of formal sums:

$$M_0 = \begin{bmatrix} p_1 + p_2 + p_3 \\ 0 \\ m_1 + m_2 + m_3 \end{bmatrix}. \qquad \diamond$$

5.2.1.3 Association of Functions to Arcs

In a colored Petri net, arc labels are no longer simple integers as in the case of ordinary Petri nets, but functions that represent integer matrices. For each color of a transition (i.e., the way to fire it), it is necessary to describe which colors of tokens will be removed from the input places (pre-condition) and which colors of tokens will be added to the output places (post-condition).

The function associated with an arc has as its domain the Cartesian product of the set of colors of the transition and the set of colors of the place (input or output of the transition, according to the arc). The image set of the function Pre is the set $\mathbb{N}$ of positive integers, which describes how many tokens of each color must be removed from the input places when the transition is fired:

$$Pre(p,t):\ C_{sc}(t)\times\ C_{sc}(p)\longrightarrow \mathbb{N},$$

and the image set of the function $Post$ is the set $\mathbb{N}$ which describes how many tokens of each color must be added to the output place when the transition is fired (see the simplified definition later):

$$Post(p,t):\ C_{sc}(t)\times\ C_{sc}(p)\longrightarrow \mathbb{N}.$$

Based on the matrices Pre and $Post$, the matrix $W(p,t) = Post(p,t) - Pre(p,t)$ is defined.

The matrices Pre, $Post$, and W are matrices of matrices because their elements (the arc labels), described by functions, are indeed matrices where the rows represent the ways to fire transitions, and the columns represent the possible colors in the places.

Example 5.4 Colored Petri net (cont.)
The colored Petri net shown in FIG. 5.4 has three functions: *part*, *machine*, and *id*. The latter does not alter the colors of the tokens.

The matrix Pre is given by:

$$Pre \;=\; \begin{array}{c} \\ P \\ O \\ M. \end{array} \begin{array}{c} \begin{array}{cc} start & end \end{array} \\ \left[\begin{array}{cc} Pre(P,start) & Pre(P,end) \\ Pre(O,start) & Pre(O,end) \\ Pre(M,start) & Pre(M,end) \end{array}\right] \end{array}$$

The function $Pre(P,start)$, which labels the arc connecting the input place P to the transition *start*, is given by the matrix $Pre(P,start) : O\times P \to \mathbb{N}$, where $O = C_{sc}(start)$ and $P = C_{sc}(P)$.

The function $Post(M,start)$, which labels the arc connecting the transition *start* to the output place M, is given by the matrix $Post(M,start) : O \times M \to \mathbb{N}$, where $M = C_{sc}(M)$. The functions $Pre(P,start)$ and $Post(M,start)$ are, respectively, described by the matrices:

$$\begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{ccccccccc} o_{11} & o_{12} & o_{13} & o_{21} & o_{22} & \cdots & o_{33} \end{array} \\ \left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 1 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 \end{array}\right] \end{array}, \qquad \begin{array}{c} \\ m_1 \\ m_2 \\ m_3 \end{array} \begin{array}{c} \begin{array}{ccccccccc} o_{11} & o_{12} & o_{13} & o_{21} & o_{22} & \cdots & o_{33} \end{array} \\ \left[\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 1 \end{array}\right] \end{array}.$$

◇

The colored Petri net primarily relies on matrix notation to make the Petri net more compact. On one hand, this notation provides a visual advantage, as the representation is quite compact. On the other hand, the complexity of the system is still that of the "unfolded" system formed by the ordinary Petri net.

A shorthand way to represent the function is to directly indicate the color to be removed (or added) from (or to) the place. In this case, the domain of the function is the set of colors of the transition, and the image is the set of colors of the place:

$$Pre(p,t) : C_{sc}(t) \to C_{sc}(p)$$

$$Post(p,t) : C_{sc}(t) \to C_{sc}(p),$$

and the name of the function is used as the label of the arc in the graph.

Example 5.5 Colored Petri net (cont.)
In the example from FIG. 5.4, the function $Pre(P, start)$ is given by *part:* $O \to P$, which labels the arc $(P, start)$. This function, applied to the colors o_{ij} from the set of colors O of the transition *start*, indicates that a token of color p_i should be removed from place P, $part(o_{ij}) = p_i$. This same function, labeling the arc (end, P), applied to the set of colors O of the transition *end*, indicates that a token of color p_i should be placed in place P. ◇

5.2.1.4 Analysis

The colored Petri net was introduced due to the difficulty of finding invariants in the predicate-transition model, a high-level Petri net that will be presented in Section 5.2.2. There are two ways to approach the analysis:

- Work with the ordinary Petri net obtained by *unfolding* the colored net, or
- Use directly the matrix of linear functions between the sets of colors.

In the first case, if the number of colors is finite, it is possible to enumerate the accessible markings. The analysis by enumeration of markings can be done exactly in the same way as for an ordinary Petri net. However, the combinatorial explosion is very large, as it is necessary to differentiate between tokens of different colors.

n the second case, there is an algorithm for searching the marking tree, based on the notion of equivalent markings. When the colors and functions exhibit a certain symmetry, it is possible to define equivalence classes among the accessible markings. It is necessary to structure the color sets with a symmetry of the rotation type. Two markings M_1 and M_2 are equivalent if and only if there exists a rotation ϕ such that $M_1 = \phi(M_2)$. The tree obtained in this way has a smaller dimension than the one obtained from the *unfolded* model.

The combinatorial explosion is thus reduced, and the analysis becomes closer to that of the *underlying* net (the ordinary net obtained by replacing the functions associated with the arcs with weights).

Regarding the calculation of conservative and repetitive steady-state components, it can be performed similarly to the method used for ordinary Petri nets.

Place invariants are obtained through a sequence of transformations aimed at reducing the incidence matrix without changing the set of invariants. The rules are based on the Gaussian elimination method. However, instead of performing integer multiplication, one must perform function composition (functions of functions). Each time a row (or column) of zeros is produced in the matrix W, a component is obtained. However, this is not always possible. Moreover, the set of components does not necessarily form a vector space.

Although there is no general algorithm to solve the matrix, the reduced matrix, obtained after applying the rules, is often such that the invariants can be found by inspection.

Example 5.6 Colored Petri net (cont.)
Returning to the previous example, two conservative components can be found. Indeed, by multiplying the second row by *part* and noting that *part(id) = part*, one obtains:

$$W_1 = \begin{bmatrix} 0 & 0 \\ part & -part \\ -machine & machine \end{bmatrix} \begin{bmatrix} P + part(O) \\ part(O) \\ M \end{bmatrix}.$$

Thus, the place invariant is found:

$$M(P) + part(M(O)) = M_0(P) + part(M_0(O)).$$

Multiplying the second row of the matrix W by *machine* and adding it to the third row yields the place invariant:

$$machine(M(O)) + M(M) = machine(M_0(O)) + M_0(M).$$

The interpretation of these invariants is straightforward using formal sums. Suppose the machining o12 o_{12} is being performed, with part p_2 and machine m_1 on hold. We have $M(P) = p_2$, $M(O) = o_{12}$ and $M(M) = m_1$. The first invariant describes the conservation of parts:

$$p_2 + part(o_{12}) = p_2 + p_1,$$

and the second describes the conservation of machines:

$$machine(o_{12}) + m_1 = m_2 + m_1.$$

◇

5.2.2 Predicate-Transition Petri Nets

5.2.2.1 Variable Notation, Semi-Unification

In the case of colored Petri nets, the descriptive power is enhanced by replacing the integers in the incidence matrix of an ordinary Petri net with functions. In predicate-transition nets, the transitions of an ordinary Petri net are viewed as rules in a propositional logic system (without variables), and the descriptive power is further increased by replacing them with rules from first-order logic (rules with variables). Thus, not only

is the representation more concise, but the model also allows for a better study of the structural and behavioral properties of the system.

A rule (transition) then describes a family of events rather than just a single event. The family is defined by the set of possible substitutions of variables with values. Instead of rules of the form *perform* operation

$$\begin{array}{c} \textit{if} \text{ part } p_1 \textit{ and} \text{ machine } M_2, \textit{perform} \text{ operation } o_{12} \\ \textit{if} \text{ part } p_2 \textit{ and} \text{ machine } M_2, \textit{perform} \text{ operation } o_{22} \\ \textit{if} \text{ part } p_i \textit{ and} \text{ machine } M_j, \textit{perform} \text{ operation } o_{ij} \\ \ldots \end{array}$$

we have rules of the form

$$\textit{if} \text{ part } <x> \textit{ and} \text{ machine } <y>, \textit{then} \text{ perform operation } <o>.$$

These variables $<x>$, $<y>$ and $<u>$ will assume values from the set of *constants* p_i describing the parts waiting, the set M_j of available machines, and the set o_{ij} of operations to be performed. These constants play an analogous role to colors in the colored Petri net, but variables are associated with the arcs of the predicate-transition Petri net, instead of functions.

In the predicate-transition model, the *unfolded* transition is viewed as a scheme that has, as instances, all the transitions (of an ordinary net) representing the same event.

Example 5.7 Predicate-transition Petri net
In Example 5.1, the transitions t_1, t_3, and t_5 (Pr_1) can be considered as *instances* of a single *transition-schema* (check if the machine is free and if the piece is waiting; if so, start the operation), as can be seen in FIG. 5.5.

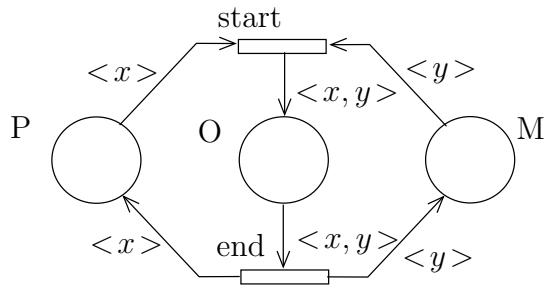


Figure 5.5: Predicate-transition net.

The places p_3, p_6, and p_9, related to Pr_1 (machines M_1, M_2, and M_3 free, respectively), can also be seen as a *place-schema* (place *folded M* in FIG. 5.5), represented by the predicate *Machine* M_i *free*. The same applies to the places p_2, p_5, and p_8 — represented by the predicate *Machine* M_i *performs operation on piece* j (folded place O) — and to the places p_1, p_4 and p_7 — represented by the predicate *piece* j *waiting* (folded place P).

The reasoning is the same for transitions and places related to process Pr_2. ◇

As can be seen in this example, the places in the net represent the dynamic relationships of the system, indicated by predicates. For this reason, the model is called predicate-transition Petri net.

It is not necessary to enumerate *a priori* all the authorized tokens in the places P, O, and M. The different ways to fire the transitions can be ignored, as it is enough to know that all elements of the same family will be treated in the same way. The rules are applied to a current marking. By searching for all possible substitutions for the variables, one knows, at a given moment, in which ways a transition can be fired. This mechanism of substituting variables with constants (tokens) is called *unification*. More precisely, it is called semi-unification, since variables are always directly substituted by constants and never by other variables.

When the unification mechanism produces more than one solution, it is said (in Artificial Intelligence terminology) that there is a conflict. The conflict space comprises all the transitions (rules) that can be fired for a given marking (fact base), and for each transition, the set of possible substitutions. This notion of conflict is not the same as the more restrictive one that concerns transitions in a Petri net (see Chapter 2, Section 2.1.7).

To select the tokens and the transitions to fire, additional firing conditions are associated with the transitions (for example, to specify that certain machines can only handle certain parts), similar to the case of interpreted Petri nets, seen in Chapter 4. Such conditions can be written as logical formulas using the variables of the input arcs, as well as the operators and predicates associated with them. However, these conditions only apply to the constants associated with the tokens that are eligible to be moved when substituted for the variables in the firing of the transition. Thus, an instance of the *scheme-transition* is generated by a consistent substitution of the variables with the individuals that make up the marking.

Similarly, the actions associated with the transitions are written using the variables associated with the input and output arcs (the arcs are labeled with formal sums of variables).

It is worth noting that in a Colored Petri net, the specific choice of a color from the set of tokens is described by functions associated with the arcs. In a predicate-transition net, these choices are expressed by conditions associated with the transitions that the variables must satisfy (except in the simple case of constant equality, which can be expressed using the same variable associated with more than one input arc of a transition). Thus, from this perspective, predicate-transition Petri nets are closer to interpreted Petri nets.

5.2.2.2 Concept of n-tuple of constants and variables

In the firing of transition *start* (FIG. 5.5), a constant representing a part is removed from place P, another representing a machine is removed from place M, and both are placed in place O, thereby encoding the fact that a part and a machine are temporarily related. At the moment of firing transition *end*, signifying the end of this relationship, the reverse is done. Instead of encoding this temporary relationship of two pieces of information, they are juxtaposed in a list, like in a *record* in programming language. The difference is that it is something temporary, created and destroyed during the firing of the transitions. In this way, a *dynamic relationship* between constants can be expressed. These lists of constants are called *n-tuples*.

After the firing of *start*, the identity of the machine used does not disappear but is maintained as an element of the n-tuple $<p_i, m_j>$ in place O, representing the dynamic relationship *part* p_i *undergoing operation on machine* Mj. In the case of the colored

Petri net (FIG. 5.4), place O contains the token of color oij.

The marking ceases to be a distribution of constants in places and becomes a distribution of n-tuples of constants. The n-tuple $<p_1, m_2>$ differs from the sum $<p_1> + <m_2>$, because in the latter case, $<p_1>$ and $<m_2>$ will be used in the transition firing independently, while the n-tuple $<p_1, m_2>$ is substituted for the variables as a whole. Therefore, the expressions associated with the arcs are also n-tuples of formal variables.

5.2.2.3 Definition of the Predicate-Transition Net (Simplified)

A marked predicate-transition Petri net is a quintuple:

$$N_{pt} =< P, T, V, Pre, Post, C_{onst}, A_{tc}, A_{ta}, M_0 >$$

where P is the set of places, T is the set of transitions, and

- V is a set of formal variables that will be replaced by constants from C_{onst} when the transition fires;
- Pre is the *preceding place* application that associates each arc with a formal sum of n-tuples of elements from V;
- $Post$ is the *subsequent place* application that associates each arc with a formal sum of n-tuples of elements from V;
- C_{onst} is a set of constants (e.g., the set $\mathbb{N}$);
- $A_{tc} : T \longrightarrow L_c(C_{onst}, V)$ is an application that associates each transition with a condition in the form of a predicate using both constants and formal variables;
- $A_{ta} : T \longrightarrow L_a(C_{onst}, V)$ is an application that associates each transition with an action in the form of a sequence of value assignments to the formal variables;
- M_0 is the initial marking, associating each place p in P with a formal sum of n-tuples of constants from C_{onst}.

Dynamic relationships appear as places in the net. Functions and relationships of the static part, or more precisely, their names, will appear in the annotations. The annotations of conditions and actions are expressions that *summarize* the affirmative sentences of natural language. Operators (function symbols) and predicates (relation symbols) form the vocabulary of the language. The language used is that of first-order predicate logic plus a class of simple algebraic expressions to denote linear combinations.

The definition of the languages L_c and L_a, which express the conditions and actions, determines the particular class of the predicate-transition Petri net.

The application Pre (and $Post$) is such that its module (the sum of all the elements without distinguishing them and developing the n-tuples) is equal to the weight of the arc in the underlying net: $\| Pre(p,t) \| = Pre(p,t)$. This means that there is no contradiction between a predicate-transition Petri net N_{pt} and its underlying Petri net R from the point of view of the number of tokens moved.

Example 5.8 Predicate-transition Petri net (cont.)
The machine handover coordination system, described by the predicate-transition net in Example 5.7, FIG. 5.5, is defined by:

- $C_{onst} = \{p_1, p_2, p_3, m_1, m_2, m_3\}$;
- $V = \{x, y\}$;
- There is neither a condition nor an action in this case;
- $$W = Post - Pre \quad = \quad \begin{matrix} \\ P \\ O \\ M \end{matrix} \begin{matrix} \begin{matrix} start & end \end{matrix} \\ \begin{bmatrix} - <x> & <x> \\ <x,y> & - <x,y> \\ - <y> & <y> \end{bmatrix} \end{matrix};$$
- $$M_0 \quad = \quad \begin{matrix} P \\ O \\ M \end{matrix} \begin{bmatrix} <p_1> + <p_2> + <p_3> \\ 0 \\ <m_1> + <m_2> + <m_3> \end{bmatrix}.$$

◇

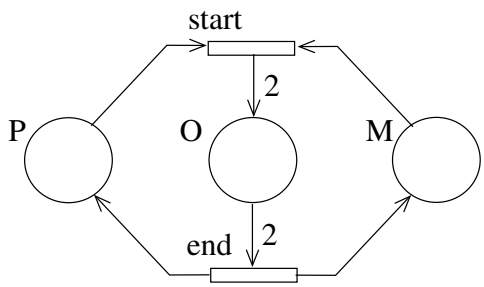


Figure 5.6: Underlying Petri net.

In the definition of a predicate-transition Petri net, the notion of an underlying net arises. This is obtained by substituting the variables with a unit weight and breaking down the n-tuples.

Example 5.9 Underlying Petri net
The net in FIG. 5.6 is the underlying net of the net in FIG. 5.5. The weight of 2 associated with the arcs $(start, O)$ and (O, end) is obtained by substituting one token for each variable in the label $< x,y >$. This net represents the *control* structure of the system, while the predicate-transition net is a structured way of describing the *control and data* set. ◇

5.2.3 Object Petri Nets

5.2.3.1 Concept of Object

The concept of an object, increasingly used in programming engineering, involves structuring an application around the entities involved, encapsulating both data structures –

in the form of an attribute list – and methods for transforming this data. This approach contrasts with the functional programming paradigm, where data structures and the functions the system must execute are defined separately.

An *object class* is defined by a set of *attributes* (also called *properties*) and a set of *operations* or *methods* that allow manipulation of the attribute values. A class can be defined based on another through *inheritance*. The new class – called a subclass – *inherits* the attribute definitions from the previously defined class. These definitions can be extended with specific attributes and operations.

Classes are merely definitions. A particular object is an *instance of an object class*. Values can be assigned to the attributes of these objects, and their operations can be executed. Encapsulation grants objects a certain autonomy and persistence. An object *is created* (instantiation) and *destroyed* (destruction) dynamically during program execution, but such events should be rare compared to the number of events corresponding to the execution of operations on the attributes.

The object-oriented Petri net can be considered a use of the predicate-transition Petri net in the context of an object-oriented approach. The tokens are no longer constants but instances of tuples of object classes. In addition to the attributes defined by the class, an *implicit* attribute contains the name of the place where the object is located. Operations are associated with transitions and correspond to preconditions (filters) that operate on the attributes of objects located in the input places, and to actions that modify these values. An operation associated with a transition t can only be executed by an object if it is located in an input place of t.

From one perspective, this approach is less structuring than the classical object-oriented approach because an operation is defined within the framework of a set of object classes (where the classes of the objects may be in the input places) rather than a single class. From another perspective, it is more structuring because it introduces a notion of *control*. For an operation to be applicable to the attributes of an object instance, it must be in a certain *state*, i.e., it must be located in a certain place.

In the example being discussed, the objects are directly associated with physical objects: parts and machines. Each of these objects can have a certain number of attributes such as:

- The name of the object instance;
- The delivery date of the part;
- The operation that must be executed on the part;
- The list of operations that a machine can perform.

5.2.3.2 Definition of Object-Oriented Petri Net (Simplified)

An object-oriented Petri net is defined by the 9-tuple:

$$N_o = < P, T, C_{lass}, V, Pre, Post, A_{tc}, A_{ta}, M_0 >,$$

onde:

- C_{lass} is a finite set of object classes, possibly organized in a hierarchy and defining a set of attributes for each class;

- P is a finite set of places whose types are given by C_{lass};
- T is a finite set of transitions;
- V is a set of variables whose types are given by C_{lass};
- Pre is the *preceding place* application that associates with each input arc of a transition a formal sum of tuples of elements from V;
- $Post$ is the *subsequent place* application that associates with each output arc of a transition a formal sum of tuples of elements from V;
- A_{tc} is a function that associates with each transition a condition involving the formal variables associated with the input arcs and the attributes of the corresponding classes;
- A_{ta} is a function that associates with each transition an action involving the formal variables associated with the input arcs and the attributes of the corresponding classes;
- M_0 is the initial marking that associates with each place a formal sum of tuples of object instances (objects must be represented by identifiers, such as their names).

Example 5.10 Object Petri Net
The graphical representation of the object Petri net model for the coordination system described in Example 5.1 is the same as in FIG. 5.5.

For this net, the following are defined:

- The set of classes $C_{lass} = \{part,\ machine\}$ with:
  - $machine = \begin{cases} name: & \text{identifier} \\ operations: & \text{list of possible operations} \\ maintenance: & \text{downtime for maintenance;} \end{cases}$
  - $part = \begin{cases} name: & \text{identificador} \\ operation: & \text{operation to be executed} \\ date: & \text{delivery date;} \end{cases}$
- The variables x, of type *part*, and y, of type *machine*: $type(x) = part$ and $type(y) = machine$;
- The places have the types: $type(P) = part$, $type(M) = machine$ and $type(< P, M >) = (part, machine)$;
- The matrices Pre and $Post$ are given by:

$$Pre = \begin{bmatrix} < x > & 0 \\ 0 & < x, y > \\ < y > & 0 \end{bmatrix} \qquad Post = \begin{bmatrix} 0 & < x > \\ < x, y > & 0 \\ 0 & < y > \end{bmatrix};$$

- An example of a condition could be:

$$A_{tc}(ini) = x.operation \in y.operations;$$

- The initial marking is defined by:

$$M_0 = \begin{bmatrix} <p_1> + <p_2> + <p_3> \\ 0 \\ <m_1> + <m_2> + <m_3> \end{bmatrix},$$

where the objects p_1, p_2, p_3 are from the class *part*, and the objects m_1, m_2, m_3 are from the class *machine*. ◇

5.2.3.3 Analysis

Predicate-transition Petri nets and object Petri nets are defined in the form of ordinary Petri nets (underlying Petri nets) equipped with inscriptions. The analysis is first performed on these nets, similar to the case of interpreted nets. The invariants of the underlying Petri net provide results on the conservation of the number of objects regardless of their classes.

If the solution to the equation $f^T W = 0$ (where $W = Post - Pre$ is the condensed incidence matrix of the system) is without variables – which is not always the case – such a solution can be used as a place invariant. To transform the matrix W in order to eliminate variables, the method of *projection* of places can be used.

Projection of Places

Consider $R = \{<a, b>, <a, c>, <b, c>\}$ as a binary relation in a set $D = \{a, b, c\}$. If we are not interested in knowing which individuals occur in the second position of the pair $<x, y>$, we can write $R = \{<a, ->, <a, ->, <b, ->\}$, or $R = \{2 <a, ->, <b, ->\}$. We can eliminate the i-th position of R, by performing a *projection* along the i-th position, denoted $|R|_i$. Thus, $|R|_2 = \{2 <a>, <b>\}$ and $||R|_2|_1 = \{3 <>\}$.

Tuples that differ only in the i-th position (located in the same place) are no longer distinguishable in their occurrences but are counted if $|\ |_i$ is applied to the place. When performing the projection of a place, all arcs adjacent to that place are considered. If the Predicate-transition net is uniform, the total projection is an ordinary net.

Performing a projection is, in a certain sense, constructing an underlying Petri net relative to only one class of object (or a set of classes). The information obtained, which is more precise than that obtained from the underlying net, approaches that obtained from colored Petri nets.

In the case of the matrices $Post$, Pre, and W, if we are interested only in a class $class_i$ of C_{lass}, all other classes are projected first, with $class_i$ being projected last. Thus, all the variables corresponding to the individuals that are not to be observed are removed from the matrix.

Example 5.11 Projection of places

In the example of the coordination system, to find the invariants related to the parts, we first perform a projection on the incidence matrix W with respect to the machines, keeping only the variables $<x>$; then a final projection is made with respect to the parts. Thus, x is replaced by "1" and y by "0". For the invariants related to the machines, the process is reversed (first the parts, then the machines).

Let be the incidence matrix W of the Petri net (Predicate-transition or Object) in FIG. 5.5, presented in Examples 5.8 and 5.10:

$$\begin{array}{c} \\ P \\ O \\ M \end{array} \begin{array}{c} \begin{array}{cc} start & end \end{array} \\ \left[\begin{array}{cc} -<x> & <x> \\ <x,y> & -<x,y> \\ -<y> & <y> \end{array}\right] \end{array}.$$

Performing the projection first on the machines, we obtain:

$$|W|_y = \left[\begin{array}{cc} -<x> & <x> \\ <x,1> & -<x,1> \\ 1 & 1 \end{array}\right].$$

And then, projecting on the parts, we finally obtain:

$$||W|_y|_x = \left[\begin{array}{cc} -1 & 1 \\ 1 & -1 \\ 0 & 0 \end{array}\right].$$

Projecting with respect to the parts and then to the machines, we obtain, respectively[1],

$$|W|_x = \left[\begin{array}{cc} 0 & 0 \\ <y> & -<y> \\ -<y> & <y> \end{array}\right] \text{ and } ||W|_x|_y = \left[\begin{array}{cc} 0 & 0 \\ 1 & -1 \\ -1 & 1 \end{array}\right].$$

Thus, in both cases, a matrix of incidence C of natural integers is obtained, and the components can be calculated as in an ordinary Petri net, using the equation $f.C = 0$. The place invariant is now a linear function not of the marking of the places, but of its projection, that is, the number of objects of class $class_i$ contained in the place.

The conservative component of the matrix obtained after projection on the part is:

$$f_{part} = \left[\begin{array}{ccc} 1 & 1 & 0 \end{array}\right]$$

and the corresponding place invariant is given by:

$$\| M(P) \|_{part} + \| M(O) \|_{part} = constant.$$

The conservative component of the matrix obtained after projection on the machine is:

$$f_{machine} = \left[\begin{array}{ccc} 0 & 1 & 1 \end{array}\right]$$

with the corresponding place invariant given by:

$$\| M(O) \|_{machine} + \| M(M) \|_{machine} = constant.$$

◇

Thus, results analogous to those found with colored Petri nets are obtained.

Another form of analysis is to introduce *error* transitions into the predicate-transition Petri net model. The design is correct if and only if the transition is non-live. Non-live transitions express invariant assertions.

[1]For simplicity, it is said that projection is performed with respect to $class_i$ when it is the final projection applied.

5.3 Characteristics of the Models

The two major advantages of Petri nets are, on one hand, their descriptive power (such as the ease of modeling parallelism and concurrency), and on the other hand, the possibility of analysis. These will be used as criteria for comparison between the HLPN models discussed in the previous section. This comparison should be made within the scope of the system to be modeled, in our case, manufacturing systems.

Although all models have the general characteristics seen in Section 5.1, each introduces them in a different way. These differences will be highlighted, and the specific characteristics introduced by each model will be presented below.

5.3.1 The Token as an Information Element

The first difference between the models concerns the way tokens are differentiated. Either each token represents an entity moving from one place to another, or it represents information that is interpreted according to the place that contains it.

The token represents *information*

In the colored Petri net model, each token is associated with a color. Thus, each place is explicitly associated with a set of colors. This color represents information. For example, in FIG. 5.4, which shows the case of a manufacturing system modeled by a colored Petri net, the colors of place M correspond directly to the identities of the machines. However, the colors of place O correspond to machining operations, meaning a combination of a part and a machine.

The token represents an *individual*

In the predicate-transition Petri net model, the token represents an individual belonging to a set of constants.

The central goal of developing the predicate-transition net is precisely the formal introduction of the concept of individuals with properties and modifiable relationships in the theory of Petri nets. This is analogous to the introduction of individuals and their structures in logic, leading from propositional logic to first-order predicate logic.

Returning to place O in the example, the tokens are n-tuples $< x, y >$ where x represents a part and y a machine. Consequently, these tokens not only retain the identity of the individuals but also clearly represent dynamic relationships between them.

In the object-oriented Petri net model, represented in FIG. 5.5, tokens are considered as individuals whose value is variable. Each token is associated with a data structure, represented by an object. Each object belongs to a class of objects to which properties (attributes) are defined. Each class can be subdivided into subclasses.

5.3.2 Folding Transitions and Places

As explained in Section 5.1, all models are based on the idea of *folding* ordinary nets. But this folding can be done with two different philosophies. In the case of predicate-transition nets and object-oriented nets, the transition is viewed as a rule, and the idea is to move from rules without variables to rules with variables, following the idea mentioned

in Section 5.2.2 of moving from propositional logic to first-order predicate logic. In the case of colored Petri nets the focus is on the matrix view of the Petri net, and thus the approach is to move from a matrix of integers to a matrix of functions, each function being represented by a matrix of integers. These points will be detailed below.

Use of Variables

In the predicate-transition Petri net, the folded place is viewed as a *place schema*, represented by a predicate $P(x)$ (or $P(x, y)$ if there is a dynamic relationship). The folded transition is viewed as a *transition schema*, represented by a rule of the type

$$IF\ P_1(x)\ and \cdots\ P_2(y)\ THEN\ Q_1(z)\ and \cdots\ Q_2(w)$$

where $P_i(\cdot)$ and $Q_j(\cdot)$ respectively indicate the predicates related to the input and output places, and x, y, z, w are the variables that label the arcs. In the object-oriented Petri net, the places belong to classes of places that indicate the classes of objects that can belong to the place. In this case, the variables belong to classes of variables that indicate the classes of objects that can replace them.

Use of Functions

In the colored Petri net model (FIG. 5.4), each place is explicitly associated with a set of colors. The same is done with each transition. The arcs, instead of being labeled by formal sums of variables, are labeled by functions of the colors of the transition to the colors of the place. This is a fundamental difference compared to the predicate-transition net.

5.4 Model Selection

A manufacturing system is primarily characterized by its ability to manufacture various types of products (manufacturing processes) simultaneously and also by its capability to rapidly modify the manufacturing plan, whether regarding the number of parts in each process or the process itself.

In a good modeling approach, the relationship between physical objects and their corresponding mathematical entities is as clear and direct as possible. This is why Petri net classes where token identity is based on the notion of *individual* are more convenient than those using the notion of color representing simple *information*. In this way, the physical elements of the system, such as parts, machines, tools, etc., are directly described through tokens.

Another reason for the interest in this representation form, still from the token perspective, is that n-tuples faithfully represent dynamic relationships existing at a given moment between the physical elements of the system.

Among the classes of nets where tokens represent individuals, the object-oriented Petri net model has a very important characteristic for this type of application: the association of a data structure with each token.

Although in the predicate-transition net it is also possible to represent a data structure through an n-tuple, this representation is more convenient in the object-oriented net. This becomes clearer when many data need to be represented about the token: the value of an n-tuple can occupy an entire row...

Furthermore, the notion of class allows one to always keep in mind the characteristics of objects, and also allows the use of only the attributes of interest at a given moment. The notion of subclass also contributes to a more concise and structured description, as well as more readable. Indeed, in a manufacturing system, each part is associated with a series of pieces of information such as: the manufacturing process to which it belongs, the operation order within the process (current or next), the machine to which it is assigned. Similarly, each machine is capable of handling a set of operations, can be free or occupied, etc.

The notion of class and subclass in the object-oriented net is also extremely useful for modeling through refinements: starting from an initial net with a certain data structure, the net and the object classes are refined in parallel.

The introduction of new parts into a manufacturing system is a good example to illustrate the suitability of the object-oriented net. In the case of the colored Petri net, it is necessary to add new colors to the color set and, therefore, modify the functions on the arcs. In the case of predicate-transition and object-oriented Petri nets, changes occur only when new parts belong to new manufacturing processes: new classes of objects (or subclasses) must be defined.

This does not mean that the conclusion will be the same for every application. For example, if the notion of routing (symmetry among colors) is evidently present within the system to be modeled, the use of the colored Petri net is recommended.

The choice of class can depend not only on the characteristics of the system to be modeled but also on the general working environment, i.e., the other tools that might be used. For instance, in manufacturing systems that use Artificial Intelligence techniques for planning and real-time scheduling control, there is an additional reason to choose either the predicate-transition net or the object-oriented net. Indeed, the philosophy of these classes is quite close to a rule-based system, unlike the colored net.

Comparison with Ordinary Petri Net

Modeling manufacturing systems to design control and supervision involves modeling the technical database part that evolves in real-time.

There are basically two options, using:

- Ordinary interpreted Petri net: the ordinary Petri net models the control, and the data structure evolves separately in parallel. The interaction between the two occurs through interpretation;
- High-level Petri net: control is modeled by the underlying net and the data structure is modeled in the token itself, through objects or constants (object-oriented net and predicate-transition net) or through colors (colored net).

Both options equally allow modeling. The choice will primarily depend on criteria such as the complexity of the control structure and the data structure. And this, from both the modeling and implementation perspectives (see Chapter 7).

5.5 Notes

For a formal definition of the colored Petri net see JENSEN (1986); for the predicate-transition net, see GENRICH (1987); for the object Petri net, see SIBERTIN-BLANC

(1985). There are other high-level Petri net models as well, such as the Petri net with individual tokens, defined by REISIG (1985).

5.6 Exercises

1. Consider the same firefighter problem from the exercise list of chapter 1, but now with four firefighters: one near the fire, one near the hydrant, and two who are on the way to assist. The synchronization mechanism is the classic *pipeline* mechanism. Model the system using: a) colored Petri net, giving the set of colors for places and transitions, as well as the functions labeling the arcs; b) object Petri net, indicating the variables on the arcs and the classes used.

2. Consider, in the lathe problem from the Exercise 4 in chapter 1 (FIG. 1.21), that there are three types of parts, with several instances of each type in the cell. Therefore, three different types of programs are needed for the lathe; the micrometer must inspect the parts according to their types. To simplify the problem, assume that the robot automatically adapts to the size of the parts. Construct an object Petri net that models this system. Suggestion: use a place that contains the programs for the lathe and for the micrometer.

Chapter 6

Petri Nets and the Representation of Time

A Petri net allows for the description of causality (one event is a consequence of another), decision, and independence between events: event a is the cause of event b, event a precedes event b, and a and b are ordered in time. It is a way to describe a partial order in a set of events and, therefore, to introduce time in a *quantitative* manner.

Several works have been carried out with the aim of explicitly using time as a continuous and quantifiable parameter [Merlin 74] [Ramchandani 74] [Molloy 81], whether for performance evaluation [Freedman 88], or for formal verification, [Berthomieu 83]. When directly associated with the Petri net, from a certain point of view, time becomes part of the control instead of being in the data part in an unstructured form, as presented in Chapter 4.1.4.

There are two major classes of models: time Petri nets and timed Petri nets.

In the time Petri net model, each transition is associated with a pair of dates $(\theta_{min}, \theta_{max})$. θ_{min} indicates the minimum duration of the transition's enabling before firing, while θ_{max} allows for calculating the maximum duration of enabling. The transition must fire within this time interval.

In the timed Petri net model, a firing duration is associated with the transitions, which are called timed transitions. Other models in the same family are nets with timed places and nets with timed arcs.

6.1 Timed Petri Net

6.1.1 Time associated with a place

If a place represents an activity A, it simply involves setting d as the duration of this activity. Thus, when the token arrives at time τ at place p, it can only leave it after $\tau' = \tau + d$ time units.

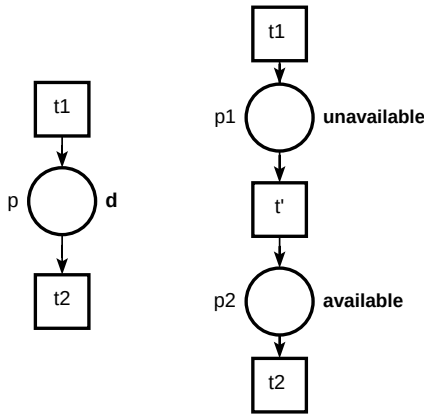


Figure 6.1: Timed place.

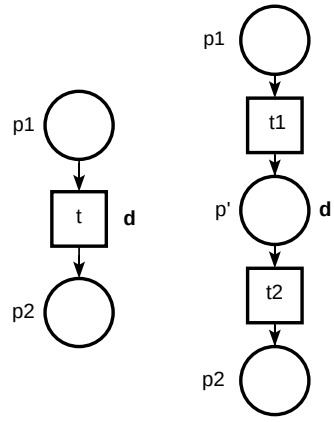


Figure 6.2: Timed transition.

The place p can unfold into a sequence *place-transition-place*, as represented in FIG. 6.1, since the two nets in the figure are equivalent, as place p_1 is substitutable (Section 3.3.1). Place p_1 corresponds to the activity being executed, transition t' corresponds to the event *elapsed time*, and place p_2 corresponds to a potential wait (synchronization with other activities) after the end of activity A. While the activity is being executed, the token cannot be used to fire the transition. The token is said to be *unavailable*. Once the activity is finished, the token becomes *available* and t_2 can eventually be triggered. Thus, the token can be in two states: available or unavailable. Only available tokens can enable the transition.

Let $M_a(p)$ be the set of available tokens in place p, a transition t is enabled if and only if $M_a(p) \geq Pre(p,t)$.

6.1.2 Time associated with a transition

The timed Petri net is obtained by associating a firing duration with each transition of the ordinary net. Its semantics is a notion of delay during which the tokens used to fire the transition are *not available* (or *not visible*) in any place. From the moment the token becomes visible in a place, it can be immediately used by any outgoing transition from that place.

Therefore, the firing is not instantaneous but has a duration. This association of time with the transition only makes sense if the transition is interpreted as an activity rather than as an instantaneous event. t is implicitly assumed that this activity is non-interruptible concerning the overall behavior of the modeled system or, alternatively, that it is simply an abbreviation.

An equivalent net can be obtained by replacing the transition with a sequence *transition-place-transition* (FIG. 6.2). The first transition $t1$ corresponds to the *instantaneous* event of the start of the activity (tokens corresponding to the arc weight are removed). The place p' serves to memorize the *activity* being executed, and the last transition $t2$ corresponds to the *instantaneous* event of the end of the activity (tokens placed in the outgoing places).

This equivalence is shown in FIG. 6.2. The firing of the transition $t1$ corresponds to the reservation of tokens (reserved tokens cannot be used to fire another transition different from t in the original net). After the firing of the transition $t2$, the tokens are

released.

Thus, tokens have two possible states: *reserved* (M_r) or *not reserved* (M_{nr}). Only non-reserved tokens can enable the transition. Therefore, the marking at any moment is given by

$$M = M_r + M_{nr}.$$

The timed Petri net can associate time either with places or with transitions. Tokens *disappear* or become *unavailable* or *reserved* for a certain period, and reappear later in the marking. FIG. 6.1 and FIG. 6.2 show that the two approaches are equivalent.

Definition:
A timed Petri net is a pair $N_{td} =< N, \Theta_f >$ where:

- N is a Petri net $< P, T, Pre, Post >$ with an initial marking M_0;
- $\Theta_f : T \longrightarrow \mathcal{Q}^+$ is the *firing duration* function, which associates each transition with a positive rational number that describes the duration of the firing.

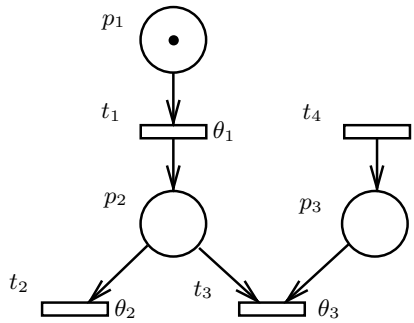


Figure 6.3: Timed Petri net.

Example 6.1 Timed Petri net
Consider the Petri net represented in FIG. 6.3, with place p_1 marked at time $\tau = 0$. Assume this net is timed, with time associated with the transitions. Each transition t_i is associated with a firing duration θ_i.

Transition t_1 is fired at time τ_1, but the token will only be visible at p_2 at time $\tau_1 + \theta_1$, thereby enabling transition t_2 at that moment. If transition t_4 (not timed) is fired at time τ_2, p_3 is immediately marked and transitions t_2 and t_3 are enabled and in conflict after time $max[(\tau_1 + \theta_1), \tau_2]$. ◇

6.2 Time Petri Net

A time Petri net is obtained by associating a enabling time interval with each transition. The firing is instantaneous, but the transition is enabled during the given time interval.

An interval $(\theta_{min}, \theta_{max})$ is associated with each transition. If the transition is enabled by the marking at time τ, it cannot fire before the date $\tau + \theta_{min}$ and must fire no later than date $\tau + \theta_{max}$.

Unlike the timed Petri net model, during the enabling of a transition, tokens continue to be "seen" by other outgoing transitions from the place.

One can associate an interval $[a, a]$ with a transition to represent a duration a. If the transition is enabled at time τ, it will fire at time $\tau + a$, provided it remains enabled by the marking at that time.

Definition:
A time Petri net is a pair $N_t =< N, I >$ where:

- N is a Petri net $< P, T, Pre, Post >$ with an initial marking M_0;
- $I(t) = [\theta_{min}(t), \theta_{max}(t)]$ is a function that, for each transition t, associates a closed rational interval that describes an *enabling duration*.

Example 6.2 Time Petri net
Consider the Petri net represented in FIG. 6.3, with place p_1 marked at time $\tau = 0$. Assume it is a time Petri. Each transition t_i is associated with an enabling duration given by the interval $\theta_i = [\theta_{min}(t_i), \theta_{max}(t_i)]$.

If transition t_1 fires at time τ_1, $\tau_1 \in [\theta_{min}(t_1), \theta_{max}(t_1)]$, place p_2 will be marked at that time. Transition t_2 can then fire at $\tau_1 + \theta_2$. If transition t_4 is fired at time τ_2, marking place p_3, transition t_3 can only fire at time $max(\tau_1, \tau_2) + \theta_3$.

The conflict between t_2 and t_3 will depend on the relationship between the times when places p_2 and p_3 are marked and the values of θ_2 and θ_3. If $\theta_3 = 0$, $\theta_2 \neq 0$ and $\tau_2 < \tau_1$, t_3 is already enabled from time τ_1, while t_2, at that time, is only enabled by the marking. Thus, t_3 fires at time τ_1 and there is no conflict between t_2 and t_3.

However, if $\theta_3 \neq 0$, a conflict exists during the intersection of the intervals $(\tau_1 + \theta_2)$ and $max(\tau_1, \tau_2) + \theta_3$. ◇

6.2.1 Representation of the *watchdog*

Certain mechanisms imply that a token should be visible only to specific transitions at a given moment. A typical case is the *watchdog*. Consider FIG. 6.4. Place *waiting* allows the system to be receptive to the arrival of an event in the form of a token in place *condition* (firing of transition *end*1). But if this place is not marked at the end of time d, an alarm is triggered (firing of transition *end*2).

The two ways of associating time in a timed Petri net (place or transition) do not allow for an accurate representation of the *watchdog* because:

- A duration of activity associated with place *waiting* would delay the firing of *end*1 even if place *condition* contained a token earlier;
- A duration of activity associated with transition *end*2 would result in the token in place *waiting* being immediately reserved (in an irreversible manner) and the alarm being triggered after time d, even if the token had, during this time, arrived at place *condition*.

The solution is given by the time Petri net, which associates an *enabling duration* $\theta(t)$ with a transition t. If this transition becomes enabled (by the marking) at time τ, it will

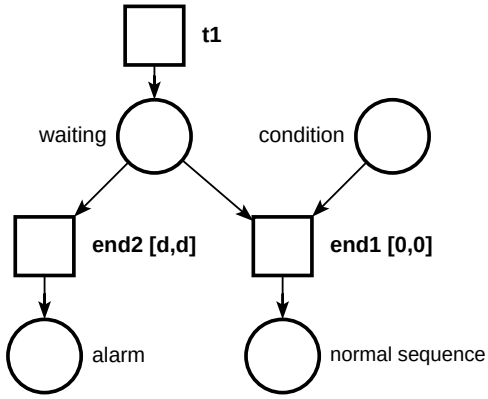


Figure 6.4: Watchdog.

remain enabled throughout the interval $(\tau + \theta(t))$, as long as the marking that enables it is not altered by the firing of another transition. The difference compared to a *firing duration* is that, throughout this interval, tokens will be available in the input places of t and can be used by a transition in conflict with t, as in the example in FIG. 6.3.

Returning to FIG. 6.4, it is sufficient to associate transition $end2$ with an interval $[d, d]$, where d is equal to the value of the watchdog, and to assign transition $end1$ a null enabling duration.

The primary domain of usage for this model (validation of communication protocols) led to the definition of an *imprecise* enabling duration for t in the form of an interval $\theta(t) = [\theta_{min}(t), \theta_{max}(t)]$. Transition t can only be fired after the time $\theta_{min}(t)$ has elapsed and cannot be fired after the time $\theta_{max}(t)$. All values of $\theta(t)$:

$$\theta_{min}(t) \leq \theta(t) \leq \theta_{max}(t)$$

correspond to enabling durations for transition t.

6.2.2 Comparison between the two models

The time Petri net model is more general than the timed Petri net model because it allows for the description of the watchdog, which the timed net does not. In this case, in FIG. 6.3, it is sufficient to use θ_2 as the duration of the *watchdog* and set $\theta_3 = 0$ so that transition t_3, which represents the end of normal operation, fires as soon as it is enabled by the marking.

To convert a timed Petri net to a time Petri net, it is sufficient to replace transition t of the former with a sequence $t_1 t_2$ with:

- $\theta_{min}(t_1) = \theta_{max}(t_1) = 0$
- and $\theta_{min}(t_2) = \theta_{max}(t_2) = \theta(t)$.

As for ordinary Petri nets (without explicit time), they correspond simply to the case where all enabling intervals are equal to $[0, \infty)$.

6.3 Stochastic Petri Net

6.3.1 Limitations of Timed and Time Nets

The use of timed Petri nets allows the construction of realistic models for evaluating discrete event systems for simulation. In fact, nothing prevents modifying the function Θ_f to associate with transitions the result of a random draw instead of a constant value. However, the properties of a timed net, or an interpreted net in general, may differ from those obtained from the underlying Petri net, as discussed in Section 4.6.

The use of the time Petri net easily describes the watchdog mechanism and the construction of a cover graph of accessible states in simple cases where the underlying Petri net is limited, and the time intervals associated with transitions are given in the form of integers. This model has been used to formally verify a certain number of protocols. However, the size of the cover graph quickly explodes when the time intervals are very different.

The main problem is that the state description of these nets must include not only the marking but also the temporal information.

To utilize the power of Markovian analysis, it is necessary for systems to be memoryless; that is, if an event causes the firing of transition t and transforms the marking M_1 into M_2, the future evolution of transitions that were enabled by M_1 before the firing of t should be identical to that which the transitions would undergo if they were enabled by M_2.

Only geometric and exponential distributions satisfy this fact. Therefore, stochastic Petri nets are those in which the enabling durations associated with transitions are defined by such distributions, to construct an equivalent Markovian process and thus analyze the net's behavior.

6.3.2 Stochastic Enabling Duration

In the case of the time Petri net, the enabling duration is a variable that assumes a value within the interval $(\theta_{min}, \theta_{max})$, with these values being equally distributed. In the stochastic Petri net, this enabling duration is a stochastic variable with an exponential probability distribution

$$Pr_{[\theta \leq \tau]} = 1 - e^{-\lambda\tau}.$$

The function $Pr_{\theta(\tau)}$ describes the probability that the firing of transition t happens before time τ and, therefore, that the enabling duration is less than this time.

Let λ be the *transition rate*; the average value of the enabling duration is given by:

$$\overline{\theta} = \int_0^\infty (1 - Pr_{\theta(\tau)}) d\tau = \int_0^\infty e^{-\lambda\tau} d\tau = \frac{1}{\lambda}.$$

Definition:
A stochastic Petri net is a pair $N_s =< N, \Lambda >$

- N is a Petri net with an initial marking;
- Λ is a function that associates, to each transition t, a transition rate $\lambda(t)$.

This is equivalent to associating to each transition a continuous enabling interval $[0, \infty)$ with an exponential distribution. This explains why the set of accessible markings is the same as that of the underlying net. The average enabling duration is $\theta_s(t) = \frac{1}{\lambda(t)}$.

6.3.3 Obtaining a Markov Chain

Consider two accessible markings M_i and M_j:

- If there is only one transition such that $M_i \xrightarrow{t_k} M_j$, the transition rate from state M_i to state M_j is given by $\Lambda(t)$;
- If there are two transitions t_k and t_m such that $M_i \xrightarrow{t_k} M_j$ and $M_i \xrightarrow{t_m} M_j$, the transition rate from state M_i to state M_j is given by $\Lambda(t_k) + \Lambda(t_m)$.

From the stochastic Petri net $N_s =< N, \Lambda >$, a Markov chain can be constructed. The states are the accessible markings of the set $A(R, M_0)$. The transition rate matrix Q is written directly from the function Λ. Column j of Q describes the evolution of the probability π_j of marking M_j over time. The element q_{ij} is the transition rate to reach state M_j from M_i. Its value is negative as it describes the transition rate that allows abandoning state M_i.

The steady-state regime of the Markov process is given by the vector Π^*, solution of the equation:

$$\Pi^{*T} Q = 0 \quad \text{with} \quad \sum_i \pi_i^* = 1. \tag{6.1}$$

Let B be a property that is verified by a set of markings EM; then:

$$Pr_B = \sum_{M_i \in EM} \pi_i.$$

This equation allows calculating, for example, the average number of tokens in a given place.

The fact that a Markov chain can be constructed from the stochastic Petri net allows for the analysis of this net. The following results are obtained:

- If the Petri net is reversible, the solution to equation 6.1 is unique;
- The set of places that belong to a positive conservative component is analogous to a closed Markov subchain;
- An unbounded place corresponds to an open chain. Due to the complexity problem, the analysis is restricted to bounded Petri nets or nets with at most one or two unbounded places (for a given initial marking);
- A particular equation of system 6.1 is described by column j of matrix Q and corresponds to the application of the cut theorem around the node corresponding to marking M_j. In the steady state, the derivative of the probability of this marking is zero.

Example 6.3 Stochastic Petri net
Consider the stochastic Petri net in FIG. 6.5, with the following transition rates:

$$\begin{aligned} \Lambda(t_i) &= \lambda_i,\ i = 1..4 \\ \Lambda(t_5) &= M(p_3)\lambda_5. \end{aligned}$$

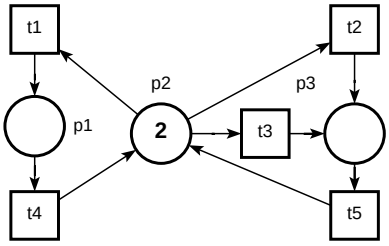


Figure 6.5: Stochastic Petri net.

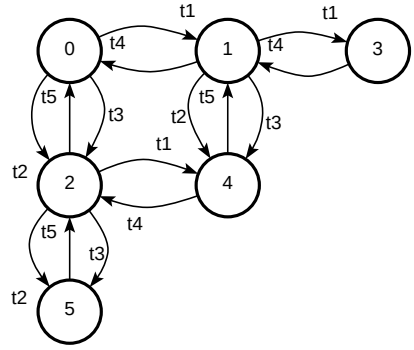


Figure 6.6: Graph $GA(R; M)$.

The graph of accessible markings is given in FIG. 6.6, with $M_0 = p_2^2$, $M_1 = p_1p_2$, $M_2 = p_2p_3$, $M_3 = p_1^2$, $M_4 = p_1p_3$, and $M_5 = p_3^2$. The transition rate matrix Q, whose rows and columns correspond to the markings M_0 to M_5, is given by:

$$\begin{bmatrix} -\lambda_1-\lambda_2-\lambda_3 & \lambda_2+\lambda_3 & 0 & \lambda_1 & 0 & 0 \\ \lambda_5 & a & \lambda_2+\lambda_3 & 0 & 0 & \lambda_1 \\ 0 & 2.\lambda_5 & -2.\lambda_5 & 0 & 0 & 0 \\ \lambda_4 & 0 & 0 & b & \lambda_1 & \lambda_2+\lambda_3 \\ 0 & 0 & 0 & \lambda_4 & -\lambda_4 & 0 \\ 0 & \lambda_4 & 0 & \lambda_5 & 0 & -\lambda_4-\lambda_5 \end{bmatrix}$$

where $a = (-\lambda_1 - \lambda_2 - \lambda_3 - \lambda_5)$ and $b = (-\lambda_1 - \lambda_2 - \lambda_3 - \lambda_4)$. The first column of the matrix Q allows us to write the equation:

$$\frac{d\pi_0}{dt} = -(\lambda_1+\lambda_2+\lambda_3)\pi_0 + \lambda_5\pi_1 + \lambda_4\pi_3$$

where π_i is the probability of the marking M_i. Note that the sum of the elements of a row in Q is zero, as the arcs leaving a state M_i are exactly the same arcs that arrive at M_j from M_i. ◇

6.4 Notes

The time Petri net was introduced by MERLIN (1974) in his doctoral thesis; the timed Petri net by RAMCHANDANI (1974) and the stochastic Petri net by MOLLOY (1981). In all these models, time is used explicitly. Works based on these models are used to evaluate the performance of systems represented by Petri nets, such as in FREEDMAN (1988), or for formal verification, as in BERTHOMIEU (1983).

For a definition of Markov chain, see CASSANDRAS (1993).

Chapter 7

Implementation Methods

All methods for implementing a design based on a Petri net can be classified according to two main sets of choices.

The first choice is whether the Petri net is implemented as a rule-based system or not. According to this choice, the implementation is done following:

- The procedural approach: this involves developing methods that allow writing programs using procedural languages such as Pascal or ADA, whose behavior is identical to that specified by the Petri net;
- The declarative approach: this considers that a Petri net is a rule-based system (see item 2.3) and bases the implementation on the use of a specialized inference mechanism called a *Petri net player*. The program that performs this inference engine does not depend on a specific a pplication. It implements the definitions corresponding to the dynamic behavior of the Petri net (enabled transition, firing of a transition, etc.).

Another fundamental choice concerning the implementation of the net will be made:

- In a centralized manner;
- In a decentralized or even distributed manner.

It is evident that these choices will be made during the final stages of design. These choices are not entirely independent, as it would not be realistic to attempt a centralized and procedural implementation of a system described by a Petri net with a high degree of parallelism (a large number of enabled and parallel transitions for a given marking).

On the other hand, in the case of decentralized implementation, it is possible to implement certain parts of the system procedurally and other parts non-procedurally.

7.1 Procedural Approach

The net describes a sequential process

The net corresponds to a conservative component that contains only one token. Therefore, it is a Petri net belonging to the subclass of state machines.

The token can be viewed as the instruction register of the sequential program that should have the same behavior.

Procedures are associated with the transitions or places, depending on whether the time is associated with the transitions or places. Both forms are equivalent.

The program is written directly; shared places correspond to *if* or *case of* or *go to* instructions.

Case of an arbitrary net

The reachability graph of the Petri net markings is computed, which is equivalent to transforming the net into an equivalent state machine. The problem is the combinatorial explosion of the number of states...

7.2 Non-procedural Approach

7.2.1 Principle

As mentioned in Chapter 2, the Petri net can be considered as a production rule system.

Each transition is then regarded as a state transformation rule; from a given state (marking), an applicable rule must be found to move to the next state.

The non-procedural approach involves writing a specialized inference engine capable of *playing* the Petri net, that is, moving the tokens in accordance with the transition firing rules. In the context of program engineering, this approach can also be considered as an *execution of the design specification*.

When defining the interpretation of a Petri net, it was seen that transitions can be associated with external events or internal events of the system being implemented.

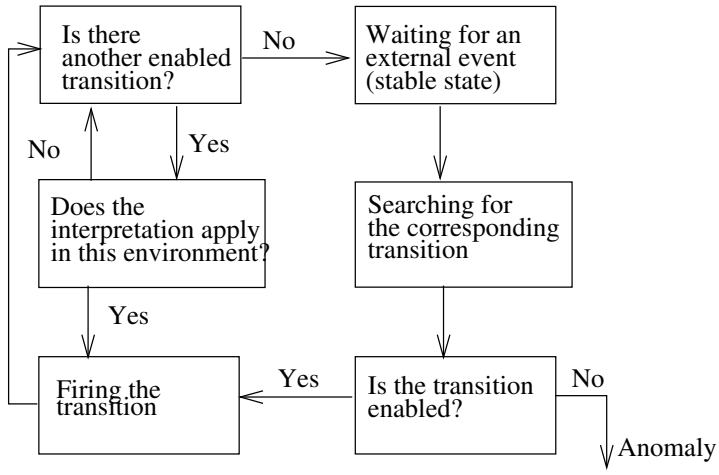


Figure 7.1: Principle of the token player.

The specialized inference engine, often called the *Petri net player*, or *token player*, has its operation described by FIG. 7.1. This must be such that the firing of external

transitions must *synchronize* with the occurrence of the associated event. This corresponds to the reception of update messages or requests from the system. On the other hand, from a given marking, all enabled internal transitions must be fired before considering the next external event. They correspond to internal conclusions (decisions) and may result in messages (commands or responses) sent to the system's external environment.

The *stable state* of a net is a marking for which only transitions associated with external events are enabled. The token player must then enter a state of "listening" to the external environment.

When a message is received, the associated transition must be found and fired. Then, from the new marking, all enabled internal transitions must be fired to reach a new stable state. By firing all internal transitions, the associated actions are executed, causing operations on data (operative part) and sending messages to the system's external environment.

7.2.2 Comparison with the procedural approach

In the non-procedural approach, an executable program is obtained that does not depend on the application, at least concerning control. Of course, there is still data processing for the operative part.

The Petri net is considered by the player as a data structure. Thus, automatic programming of the control part is achieved, hence the analogy with the execution of specifications.

The complexity of this approach varies little with the number of places and transitions of the Petri net. It mainly depends on the number of transitions and tokens involved in conflicts. Procedural-type implementations have a complexity that grows with the size of the Petri net. This approach is preferable in simple cases. In complex cases and when response time constraints are not critical, the non-procedural approach is preferable.

Another point to highlight is that, being close to Artificial Intelligence techniques, it can be used together with the player mechanism. Two cases are of interest:

- Conflict resolution;
- Diagnosis in case of anomaly.

The first case involves having an effective strategy for choosing a transition (rule) and a tuple of tokens (facts) during forward chaining operation, which corresponds to the firing of an internal transition. One can prioritize response time and organize the data and search procedures so that the first solution found is the best. Alternatively, the entire conflict space can be built and external selection rules (possibly a human operator) can be used to reach a final decision.

The second case arises from the following problem. The player mechanism is an effective tool for detecting any inconsistency between the reception of an update message and the last known state, but it does not offer an incident diagnosis tool. If the Petri net is stored as a data structure operated on by the player, it can also be considered as a semantic diagram for another deduction system responsible for diagnosis. The Petri net is thus a convenient way to know the presumed state of knowledge (the marking) as well as all coherent possibilities for its evolution (the sequences of transitions).

7.3 Decentralized Approach

The system design must have been done using the notion of object, or better yet, sequential processes as structuring elements. This approach implies decomposing the Petri net into a number of processes that will be implemented as sequential processes communicating with each other. These sequential processes will be implemented by tasks executed in parallel on a processor or, in a distributed manner, on a local network.

The objects that are sequential processes can be implemented directly, while others will need to be decomposed more finely, transformed into finite state machines, or implemented non-procedurally.

Communications between tasks are based on the classic principles of:

- Asynchronous communication;
- *Rendez-vous*;
- Remote procedure execution.

But, for the implementation of these mechanisms, new choices are necessary, and according to these choices, constraints will arise. These choices and constraints are obviously closely linked to the chosen implementation language and the multi-tasking operating system (or the communication primitives offered by the local network) of the target computing system.

One of the important issues concerns the mutual nominal knowledge that the communicating tasks must have or not. If tasks A and B must communicate or simply synchronize, there are four possibilities:

1. A knows B, and B does not know A;
2. B knows A, and A does not know B;
3. A and B mutually know each other;
4. A and B do not know each other, but:
    - They know a global object specific to this exchange;
    - They are known by a global object, the *configurator*, having knowledge of the exchange and the ability to facilitate its execution.

Cases 1 and 2 are different in the case of asymmetric communication such as asynchronous communication.

To increase the autonomy of objects and promote their reuse in another part of the application or even in another project, it is always beneficial for objects to nominally know a minimum number of other objects. Thus, the choice of solution 3 is of no interest. On the other hand, solution 4 is interesting but sometimes difficult to implement.

Chapter 8

Petri Nets, Non-Classical Logics, and Hybrid Systems

8.1 Fuzzy Petri Nets

Diagnostic and recovery procedures for discrete event systems, such as manufacturing systems, involve reasoning about objects and resources and their state changes. Additionally, it is often necessary to deal with data that may be vague or uncertain in nature. Classical logic is not very suitable for such tasks, especially when dealing with changes, as frame and ramification problems make any reasoning difficult. On the other hand, Petri nets are well-suited to represent states of dynamic systems composed of concurrent processes that share resources, but they are not adequate for dealing with system uncertainty, if it exists.

The theory of fuzzy Petri nets is recent, and the majority of work has been published in technical journals and conferences.

8.1.1 Requirements for Models of Dynamic Systems

The development of computational technology has enabled a variety of methods, techniques, and tools to solve problems associated with the design and operation of dynamic systems. An important class of such systems are those driven by instantaneous events, the discrete event systems, called DES.

There are two fundamental aspects in such systems: evolution over time and uncertainty concerning the information they handle. In general, on one hand, works on the representation of the evolutionary aspect rarely consider the incomplete or uncertain aspect of the world's description and its evolution. On the other hand, the dynamic aspect of information is mostly not considered in works dealing with the uncertainty aspect. These aspects are intrinsically multidisciplinary, utilizing concepts from Software Engineering, Systems Analysis, Artificial Intelligence, Control, and Databases. Various models have been developed, but none are general-purpose, as none consider all aspects involved in the problem.

8.1.1.1 Petri Nets

The dynamic aspect of a DES is captured by the Petri net in a formal and at the

same time natural way. As we have seen in the first part of this book, this model allows representing true parallelism, concurrency, precedence constraints, etc. The analysis of so-called good properties (such as boundedness and liveness) and structural properties (place and transition invariants) are an efficient aid during the specification and design phases in the lifecycle of a DES, as seen in chapter 3. The obtained specification can also be simulated and directly implemented by a Petri net *player*, as seen in chapter 7.

An ordinary Petri net can only represent whether the system is (or is not) in a given state. When the system is complex and consists of a set of processes, it may be necessary to use high-level Petri nets (Chapter 5). In both cases, only precisely known information is considered.

If the Petri net is a model of a physical system and is implemented in real-time concurrently with its evolution, transitions are only fired when it is certain that the associated event actually occurred (Chapters 4 and 7). If the event associated with the transition occurs and the transition is not enabled by the marking, an inconsistency in the model is detected. It is possible to represent time and communication with the external environment, but it is not possible, however, to represent incomplete information, such as "the operation duration is *approximately* 5 minutes". [1]

8.1.1.2 Fuzzy Sets

Fuzzy sets were introduced by Lotfi Zadeh from the University of Berkeley in 1965. These sets allow dealing with the representation of classes whose boundaries are not well defined, through characteristic functions that take values in the interval $[0, 1]$. Fuzzy sets are a powerful tool for knowledge representation because they allow considering vague or flexible categories used by experts in rules such as "*if X is A then Y is B*," where A and B can be fuzzy sets. This type of rule is used in fuzzy control systems and fuzzy expert systems.

8.1.2 Combining Petri Nets and Fuzzy Sets

8.1.2.1 The Various Approaches

Several authors have demonstrated the relationships between Petri nets and logic programs, such as MURATA (1988), and Petri nets and Production Rule Systems, such as VALETTE & ATABAKHCHE (1987) and ZISMAN (1978). It is natural, therefore, to introduce fuzzy logic into the Petri net model. Since the pioneering work of LOONEY (1988), various authors from the Petri net and Artificial Intelligence communities have proposed different types of fuzzy Petri nets (FPN). Despite sharing the same name, these models are based on different notions and are more or less consistent with Petri net theory.

8.1.2.2 Tokens and Markings

Although not formally defined in any paper, one can imagine that fuzzy markings could be defined by associating a fuzzy number with each place, indicating the number

[1] Incomplete information can be expressed in terms of vagueness and imprecision. Generally, *vagueness* is associated with the difficulty of making rigid or precise distinctions; *imprecision* is associated with *one-to-many* relationships.

of tokens in the place. However, this must be consistent with place invariants, otherwise, the number of markings would grow infinitely (when imprecision increases), resulting in an unbounded system.

For example, in the FPN model of CAO & SANDERSON (1993), fuzzy marking is assigned to a place to quantify the uncertainty of the existence of a token there. In CARDOSO (1991), fuzzy marking (of an object Petri net) is defined by assigning each token its fuzzy location, characterized by a fuzzy set of places (the places where the token may be). In other works where a fuzzy logical proposition is associated with a place, a membership function is associated with the token. The token's value indicates the truth degree of propositions and changes with the firing of transitions, as in the works of CHEN (1990) and SCARPELLI (1996).

8.1.2.3 Transitions

Although in the classical Petri net model tokens are removed when transitions fire, this is not always the case when the Petri net is used with Artificial Intelligence techniques. In logic, for example, a proposition (the antecedent of a rule) remains true after the rule is fired, so the token in the corresponding place is not removed, as noted by Murata (1988). In other works, such as Cardoso (1991), the tokens represent entities, and the places are the possible states of these entities. If a token f is in a place p, it means that the proposition "token f is in state p" is true.

In cases where fuzzy Petri nets (FPNs) represent fuzzy production rules, the tokens are removed. This does not imply that the proposition becomes false but rather that the corresponding information has been used in carrying out a proof scenario.

Most works include a certainty factor, confidence, or truth value associated with the transition. Since a transition in an FPN corresponds to a rule in a production system, firing a sequence of transitions (considering the confidence factor of each) defines a fuzzy firing sequence that is more or less likely to be fired.

Another approach is to associate an authorization function with the transition: a *pseudo* firing occurs when there is imprecise or uncertain information. During normal operation, the FPN behaves like a classical Petri net. A generalization of this work is the association of a fuzzy date with each transition. This date is compared with the current time to verify whether the transition has *not yet* been fired, or if it has *definitely* been fired, or if it is *possible* that it has been fired (*possible* in the sense of Possibility theory).

8.1.2.4 Considerations on Consistency

The Petri net is characterized by its dynamics: it is possible to *execute* a net to analyze the behavior of the modeled system. The way membership functions of fuzzy sets are assigned to tokens (or places) and certainty factors are associated with transitions will only be promising if it is consistent with Petri net theory.

An important point in Petri net theory is that through linear programming (Chapter 3) it is possible to extract specific subnets. A place invariant is a subnet in which the total number of tokens in its places is constant. A transition invariant is a subnet in which the transformation of markings obtained by firing its transitions is null (the final marking is equal to the initial marking). It is, therefore, important to explore at least one of these notions. Thus, the use of transition invariants allows finding, in Petri nets modeling

logical programs, an *optimal* proof: a proof in which all intermediate deductions are indeed necessary to obtain the conclusion (which is not always the case in general). In the case of FPNs modeling physical processes, for each token (which is an object instance in an object-oriented Petri net) there is a place invariant that contains it with *certainty* (the location of the token may be imprecise or fuzzy, but it is contained in the place invariant).

Representing rules by transitions in a Petri net is an old idea, but it remains an open question and is certainly unresolved when fuzzy rules are considered. This is a direct consequence of tokens being consumed and produced by firing transitions in a net, whereas in classical logic, due to the property of monotonicity, a deduction always adds new truth values without falsifying those already used (see the discussion on linear logic in the next section).

As seen in chapter 2, the grammar $S =< I\!P, Q >$ associated with the net R is defined by its vocabulary $I\!P$ and the set Q of rewriting rules $t_i : \mu(Pre(., t_i)) \rightarrow \mu(Post(., t_i))$. An initial marking M of the net corresponds to an axiom $\mu(M)$ of the grammar, from which new words are deduced.

Using this notation, consider, for example, the rewriting rule $\mu(t) : P \rightarrow Q \wedge R$, whose Petri net is represented in FIG. 8.1.a. This is quite similar to the propositional logic formula $P \rightarrow Q \wedge R$. This formula is equivalent to $(P \rightarrow Q) \wedge (P \rightarrow R)$ or $P \rightarrow Q$ and $P \rightarrow R$. Now consider rewriting rules similar to these two formulas, we have $\mu(t_1) : P \rightarrow Q$ and $\mu(t_2) : P \rightarrow R$ whose net is represented in FIG. 8.1.b. In classical logic, these two formulas are equivalent because both lead to the same conclusion: if the fact P *is true*, one can deduce that Q and R are true. But the two Petri nets in FIG. 8.1 are different. In FIG. 8.1.a both places Q and R are marked, whereas in FIG. 8.1.b only one of these places will be marked.

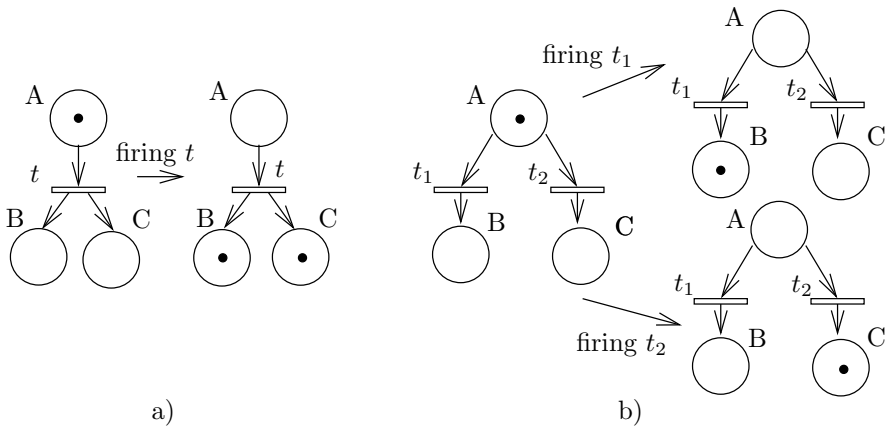


Figure 8.1: Petri net and logic: a) $P \rightarrow Q \wedge R$, b) $(P \rightarrow Q) \wedge (P \rightarrow R)$

In general, the application of FPNs, specifically in the context of control and supervision of manufacturing systems, can follow two lines according to their use as:

- A model for reasoning: the transitions represent diagnostic rules, each sequence of transition firings represents an explanation that can be used for error recovery. The FPN as a whole represents the diagnostic expert system;

- A model of the physical system: the transitions represent possible state changes; the sequences of transitions represent possible behaviors. The FPN is a tool to update the system state at the supervision level, using *poorly* known information (with imprecision or uncertainty), without obtaining inconsistent states.

8.2 Petri Nets as Semantics for Linear Logic

As seen in the previous section, a problem with representing the Petri net using classical logic is that it does not allow working directly with the notion of resources. Linear logic, however, does, and thus the Petri net can be used directly to provide semantics to linear logic.

Besides the notion of resources, another motivation for using linear logic is the more precise representation of a firing sequence. In fact, a problem in Petri net theory is that the characterization of the marking through a vector M is *complete and precise*, which is not the case for the sequence s, as seen in section 2.1.8. The existence of a characteristic vector $\mathbf{s}$ that is a solution to $M' = M + C.\mathbf{s}$ does not guarantee that the sequence s can be effectively fired. Besides disregarding the order of the sequence, only the transformation of the marking is described.

8.2.1 Linear Logic: Basic Concepts

Linear logic, proposed by Girard (1987), is of great interest when one wishes to manipulate consumable resources; it differs from classical logic because two rules are not valid: the contraction rule and the weakening rule. Linear logic has three sets of connectives:

- Multiplicative connectives:
  - $\otimes$, called *times*, expresses AND of resources,
  - $\multimap$, called *linear implication*, expresses causal dependency between resources,
  - $\parr$ called *par*, when used with negation, expresses precedence notions;
- Additive connectives &, called *with*, and $\oplus$, called *plus*, which allow expressing internal and external choice;
- Exponential connectives **!**, called *of course* and **?**, called *why not*. **!**A indicates that the resource A can be *used* as much as needed (it is unlimited) and **?**A indicates that the resource A can be *produced* as much as needed. These connectives allow reintroducing the notions of contraction and weakening, if necessary, but for specific formulas and in a controlled manner.

Linear logic also handles negation but with an interpretation different from that of classical logic: linear negation is denoted $^{\perp}$ and does not express truth/falsity, but concepts of action/reaction, such as production and consumption.

Thus, in linear logic, if A and B are resources, $A \otimes B$ represents a resource consisting of A and B simultaneously, and $A \multimap B$ represents the action of using A to produce B (causal dependency).

When resource A is available, it is possible to obtain resource B through the sequent calculation between the formulas A and $A \multimap B$. Unlike in (monotonic) classical logic, from this point onward, resource A is no longer available; the deduction represents the consumption of this resource and the production of B.

Therefore, well-formed formulas describing the state of resources are consumed when used in the proof of a sequent (when an idle resource is allocated to a task, it becomes occupied).

8.2.2 Description of Petri net using linear logic

Any Petri net can be described by a collection of well-formed formulas of linear logic: markings are described by consumable formulas and transitions by non-consumable formulas (using the **!** connective). A formula like $A \otimes D$ describes the distribution of tokens (marking) such that places A and D each contain one token. This marking can be a partial marking. This formula is correct because the tokens can only be used once.

In turn, a transition t with $Pre(t) = \{A\}$ and $Post(t) = \{B, C\}$ is represented by the formula

$$!(A \overset{t}{\multimap} B \otimes C).$$

Since transitions are not consumed by firing (if a new token arrives at the input place, it can be immediately fired), it is necessary to use the **!** connective. However, in a proof, the **!** connective does not appear since the transition is fired only once for a given marking.

A collection of linear logic formulas corresponding to a description of a Petri net with an initial marking will take the following forms:

- For the initial marking

$$\bigotimes_{A_k \in \text{set of marked places}} m_k.A_k; \tag{8.1}$$

  Here A_k denotes a token in place A_k, and $m_k.A_k$ signifies that there are m_k tokens in A_k.

- For each transition t_j of the net:

$$! \left(\bigotimes_{\substack{A_i \in \text{input places} \\ \text{of } t_j}} m_i.A_i \multimap \bigotimes_{\substack{A_o \in \text{output places} \\ \text{of } t_j}} m_o.A_o \right). \tag{8.2}$$

8.2.2.1 Transition firing

A transition t of a Petri net is enabled if $M \geq Pre(., t)$, where M is the marking vector and $Pre(., t)$ is the input function. The firing consists of removing tokens from the input places and placing them in the output places using the equation

$$M' = M + C.t$$

where $C = Post - Pre$ is the incidence matrix.

Firing a transition corresponds in linear logic to proving the sequent:

$$\frac{M,\ t}{M'}$$

where the formula M is the initial marking in the form of equation 8.1 and the formula t is the transition in the form of equation 8.2. One can also write this sequent in the form: $M,\ t \vdash M'$.

Firing is only possible if all atoms on the left side of the formula t are present in the formula M. In the case of the transition t treated above, if the marking of the net is A, then we have

$$\frac{A,\ (A \overset{t}{\multimap} B \otimes C)}{B \otimes C}. \tag{8.3}$$

The resource A was consumed, and the resource $B \otimes C$ was produced. In terms of Petri nets, this corresponds to removing tokens from the input places of t and placing them in the output places of t.

8.2.3 Firing sequence in linear logic

In this section, the method by which a firing sequence can be characterized by a well-formed formula of linear logic will be presented. This will then be compared with the sequence obtained through the marking transformation calculation used in Petri net theory. The characterization of a sequence s in a Petri net is $C\mathbf{s}$, which is used in the fundamental equation

$$M' = M + C\mathbf{s}. \tag{8.4}$$

Each component $s(t)$ of $\mathbf{s}$ indicates the number of times the transition t fires in the sequence s. The vector $\mathbf{s}$ does not indicate the order of firing of the transitions, and even a solution $\mathbf{s} \geq 0$ does not imply the existence of such a sequence (necessary but not sufficient condition), as seen in Section 2.1.8.

8.2.3.1 Ordered sequences

If the sequence $t_1 t_2$ is an ordered sequence, then t_1 must fire before t_2. All tokens produced by t_1 and that can be consumed by t_2 must indeed be consumed. This means eliminating as much as possible the set of propositions that are members of the right part of the formula representing the first transition of the sequence, and the left part of the formula representing the second transition of the sequence. This corresponds, in Petri nets, to the definition of firing: all tokens present in the input places of t that enable it must be removed, and the enabling condition is the minimum marking that allows the firing. By doing so, we have both the marking transformation and the minimum marking required to fire the sequence.

Example 8.1 Firing sequence
Consider the Petri net in FIG. 8.2. Its structure (the initial marking will not be used

here) corresponds to the following set of linear logic formulas:

$$\begin{aligned} t_1 &: \mathbf{!}(A \overset{t_1}{\multimap} B) \\ t_2 &: \mathbf{!}(B \otimes \alpha \overset{t_2}{\multimap} C \otimes \beta) \\ t_3 &: \mathbf{!}(C \otimes \beta \overset{t_3}{\multimap} D \otimes \alpha). \end{aligned}$$

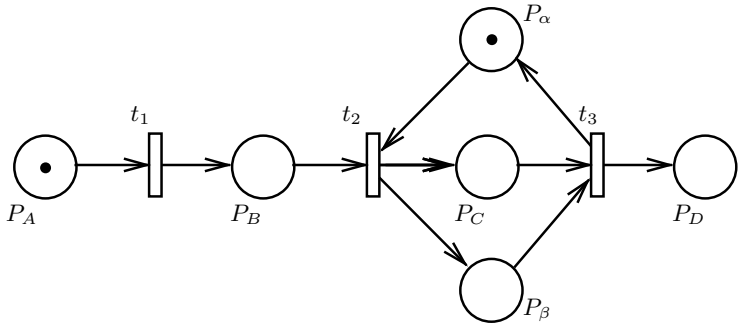


Figure 8.2: Resources and sequence.

Consider, for example, the sequence $s_1 = t_1\, t_2\, t_3$. We calculate the sequent between the formulas of t_1 and t_2, and from this with t_3:

$$\frac{\frac{(A \overset{t_1}{\multimap} B),\ (B \otimes \alpha \overset{t_2}{\multimap} C \otimes \beta)}{(A \otimes \alpha \multimap C \otimes \beta)},\quad (C \otimes \beta \overset{t_3}{\multimap} D \otimes \alpha)}{(A \otimes \alpha \overset{s_1}{\multimap} D \otimes \alpha)}.$$

The resulting formula

$$s_1 \ : \ \mathbf{!}(A \otimes \alpha \multimap D \otimes \alpha) \tag{8.5}$$

represents the sequence $s_1 = t_1\, t_2\, t_3$ after proving the sequent above.

Now consider the sequence $s_2 = t_3 t_1 t_2$. It is represented by the formula

$$\frac{\frac{(C \otimes \beta \overset{t_3}{\multimap} D \otimes \alpha),\ (A \overset{t_1}{\multimap} B)}{(C \otimes \beta \otimes A \multimap D \otimes \alpha \otimes B)},\quad (B \otimes \alpha \overset{t_2}{\multimap} C \otimes \beta)}{(C \otimes \beta \otimes A \multimap D \otimes C \otimes \beta)}.$$

The resulting formula

$$s_2 \ : \ \mathbf{!}(C \otimes \beta \otimes A \multimap C \otimes \beta \otimes D) \tag{8.6}$$

represents the sequence $s_2 = t_3 t_1 t_2$ after proving the sequent above. ◇

In both sequences, the obtained formulas represent the minimum marking required for the sequence and the marking resulting from applying the sequence to this minimum marking (in the same way as the incidence matrix).

According to the definition of Petri net, a sequence is an ordered set of transitions. Therefore, in terms of linear logic, the formula associated with a sequence has the same definition, but formulas representing transitions arranged in any order can be associated with different sequences. If s_1 and s_2 are valid sequences, the formulas $\mathbf{!}(s_1)$ and $\mathbf{!}(s_2)$ are valid.

8.2.3.2 Resources

The first point to observe, considering the sequence s_1 in Example 8.1, is that the atom α remains present on both sides of the linear implication and cannot be simplified.

However, the atom β does not appear because, in the firing of the sequence s, the resource α is consumed before being produced, while β is produced before being consumed. Formula 8.5 shows both the transformation of the marking (a token moving from P_A to P_D) and the minimum marking necessary to enable the firing of the sequence (marking $A \otimes \alpha$ in this example). As for sequence s_2 (formula 8.6), the minimum necessary marking is $(C \otimes \beta \otimes A)$. Therefore, for the marking indicated in FIG. 8.2, the firing of this sequence is not possible.

Thus, we have both the transformation of the marking and the minimum marking necessary to fire the sequence, while $C\mathbf{s}$ indicates only the transformation of the marking caused by the sequence s and does not even guarantee its existence (the component relative to place α is null).

The characterization of a sequence in linear logic depends on the firing order. Thus, sequences $s_1 = t_1\, t_2\, t_3$ and $s_2 = t_3\, t_1\, t_2$ are not represented by the same formula. The first is represented by equation 8.5 and the second is represented by equation 8.6. In Petri net theory, both are represented by the same characteristic vector $\mathbf{s} = [\,1\ 1\ 1\,]^T$, which, applied in equation 8.4 with the marking from figure 8.2 (places P_A and P_α marked), provides the same marking (places P_D and P_α marked). However, it is easy to verify that sequence s_1 can be fired for this initial marking, while sequence s_2 cannot be fired.

8.3 Petri Nets for Hybrid Systems

8.3.1 Hybrid Production System

In general, the model of a dynamic system is hybrid if the state variables are of two types: continuous (real values) and discrete (finite set values, such as integers). As seen in Chapter 1, considering the nature of the state variables and time, there are four types of models:

- Continuous system: continuous time and continuous state variables;
- Discretized system: discrete time and continuous state variables;
- Discrete system: continuous time and discrete state variables;
- Discrete event system: discrete time and discrete state variables.

In the case of hybrid systems, the state variables, as well as time, can be of both types (discrete and continuous). Thus, time is also hybrid, which implies two types of models: a continuous model and a discrete event model.

8.3.1.1 Hybrid Manufacturing Process

Hybrid models are often necessary in batch-type production systems. For example, in a manufacturing system, at the local control level, the regulation systems of motors (axis control) impose continuous models (or discretized if the control is numerical). But

the sequencing of operations and system control is based on a discrete event model. However, these two views should not be combined into a hybrid system, as they are rarely connected by dependency relationships.

In the case of an industrial process that handles a continuous flow of raw material operating in a steady state, such as an oil refinery, the simplest mathematical model is the continuous one, consisting of algebraic equations. But if there is interest in the startups and shutdowns of the facility, it is necessary to introduce differential equations to represent the dynamic behavior. It may also be necessary to introduce discrete variables to describe the different phases of the transient regime and the associated controls (opening and closing valves, heating, etc.). In the case of modeling the behavior of opening a valve for stability analysis, a hybrid model with continuous time is required.

Hybrid production systems, such as *batch systems*, represent another case. These are industrial processes that transform continuous raw material and consist of two types of equipment:

- A first type, for example, heat exchangers used to heat or cool the raw material, operates continuously. To be effective and ensure the raw material's temperature is correct, the flow through it must be constant in a steady state. During a time interval dt, a volume dV of material undergoes complete physical-chemical transformation. For example, its temperature changes from θ_0 to θ_1;

- A second type of equipment, for example, reactors, operates discontinuously. After a filling phase, an entire batch of material undergoes transformation. If a reactor is used to heat a batch of volume V, during the time interval dt the entire volume V changes from a temperature θ to $\theta + d\theta$.

Batch-type systems generally impose the use of a hybrid model. The manufacturing recipe that describes the sequence of transformations to be performed on the equipment is naturally a discrete event model (like a manufacturing process in an industrial setting), but the unit *transformed material* is not a batch of parts but a quantity of raw material that can vary according to the reactor size, and is not physically delimited in the continuously operating equipment. At all times, it must be verified whether the equipment required to operate in a steady state is properly supplied (raw material at the input and available s pace at the output), which imposes a material balance (equations involving continuous variables). Therefore, it is difficult to avoid the simultaneous manipulation of discrete states and events to describe the sequence of phases and continuous variables taking their values in the set of real numbers to describe volumes, molar masses, chemical compositions, etc.

8.3.2 Modeling Techniques

The challenge presented by hybrid systems is that, from a mathematical perspective, calculations on integers (and therefore discrete variables) are very different (generally more complex) than calculations on real numbers. There are no derivatives, and often only costly enumerative procedures exist.

The developed approaches can be grouped into four classes: continuous models with added discrete variables; discrete models with added continuous variables; models

integrating continuous and discrete within a single formalism or within two communicating formalisms.

These approaches will be illustrated with an example of a reactor that needs to be filled and emptied (FIG. 8.3).

8.3.2.1 Extended Continuous Model

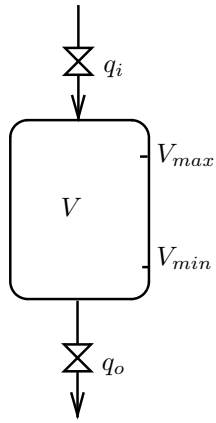


Figure 8.3: Reactor

In the case of the reactor in FIG. 8.3, the continuous model is a simple differential equation. The volume V (the only state variable) contained in the reactor is calculated from the input flow rate q_i and the output flow rate q_o (considered here as parameters) by:

$$\frac{dV}{dt} = q_i - q_o. \tag{8.7}$$

If the model is used in real-time on a processor to calculate the volume at a given instant, then the discretization instants t_n must be considered, and we obtain (assuming that the variations of q_i and q_o between two sampling instants are negligible):

$$V(t_{n+1}) - V(t_n) = (q_i - q_o)(t_{n+1} - t_n). \tag{8.8}$$

There are two valves (input and output) in the system shown in FIG. 8.3; since each can be open or closed, there are four operating configurations in this system. Equation 8.7 describes the system's behavior when both valves are open. To describe the four configurations in a single equation, equation 8.7 is augmented with two Boolean variables b_i and b_o, resulting in the equation:

$$\frac{dV}{dt} = b_i \cdot q_i - b_o \cdot q_o. \tag{8.9}$$

The Boolean variable b_i corresponds to the predicate "input valve open" (it is 1 when the input valve is open and 0 when it is closed), and the Boolean variable b_o corresponds to the predicate "output valve open" (it is 1 when open and 0 when closed).

Equation 8.9 is valid for the four possible phases of the system, thus allowing the simulation of the system's overall behavior (continuous and discrete parts in the same model):

- The reactor can be isolated: $b_i = 0$ and $b_o = 0$;
- The reactor is being filled: $b_i = 1$ and $b_o = 0$;
- The reactor is being emptied: $b_i = 0$ and $b_o = 1$;
- The reactor is being filled and emptied simultaneously (acting as a storage device to decouple two continuously operating pieces of equipment): $b_i = 1$ and $b_o = 1$.

There are three state variables (V, b_i, and b_o) and two parameters (q_i and q_o). The variable V describes the continuous part of the state, while the other two correspond to the discrete part of the state. They encode the four phases and make equation 8.9 valid regardless of the reactor's state. However, this equation does not describe the events that trigger the transition from one phase to another. It is necessary to add equations that calculate the Boolean variables based on the thresholds of the continuous variable V:

$$\begin{array}{ll} \text{If } (V < V_{max}) \text{ then } b_i = 1 & \text{If } (V \geq V_{max}) \text{ then } b_i = 0 \\ \text{If } (V > V_{min}) \text{ then } b_o = 1 & \text{If } (V \leq V_{min}) \text{ then } b_o = 0. \end{array}$$

This solution is only feasible if the system has few phase changes. Additionally, a first problem arises when the direction of the variation of the continuous quantity affects the calculation of the Boolean variables, for example,

$$\begin{array}{lll} \text{If } (V & = & V_{high}) \text{ when } V \text{ is increasing, then } b_i = 0 \\ \text{If } (V & = & V_{medium}) \text{ when } V \text{ is increasing, then } b_i = 1. \end{array}$$

A second problem arises when there are several "filling" phases in a manufacturing recipe. For instance, when it is necessary to initially put 1000 liters of product A into the reactor, and then add 500 liters of product B. The Boolean variables and thresholds quickly increase.

8.3.2.2 Extended Discrete Event Model

Discrete event models (finite automata or Petri nets) are based on discrete states and discrete time. Time is made explicit in the form of a sequence of events that are state changes (the *next state* function in an automaton, or the firing of a transition in a Petri net).

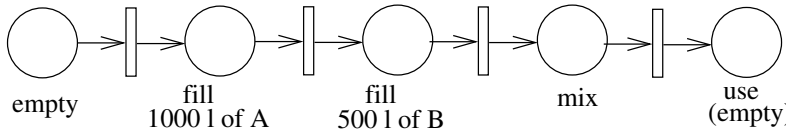


Figure 8.4: Manufacturing recipe.

A discrete event model is particularly well-suited to describe the succession of phases of a hybrid system necessary for the manufacture of a product (*recipe*). In the case of

the reactor, an elementary recipe can be described by the Petri net in FIG. 8.4. Starting from the empty reactor, 1000 liters of product A are added and then 500 liters of B are added. These are then mixed, and finally the product is used. Each phase is a place in the net.

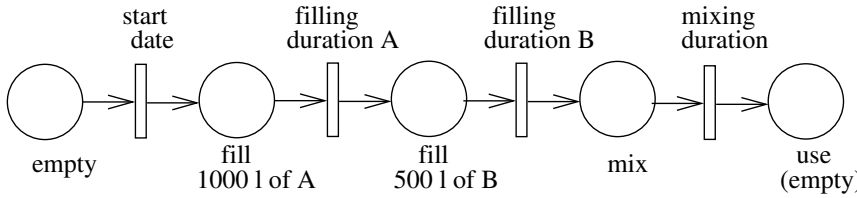


Figure 8.5: Time specification.

Introducing time into the model in FIG. 8.4 means moving from a discrete event model to a discrete model (discrete states, continuous time). In the case of the temporal Petri net, a firing duration is associated with the transition.

Associating a filling duration of the reactor with product A to the transition exiting the place *fill 1000 l of A* forces the tokens circulating in the net to remain in the place for the specified time (Petri net in FIG. 8.5). In simple cases, this duration can be calculated as $\frac{1000}{q_{i_A}}$ where q_{i_A} is the input flow rate. The value of V can be calculated at any time; thus, through simple interpolation, the hybrid state of the reactor (phase and V) can be reconstructed at each moment (continuous time).

The hybrid model is then formed by a discrete model (Petri net) complemented with a continuous variable: time. All other continuous state variables must be linear functions of time, calculable at any moment from the discrete state.

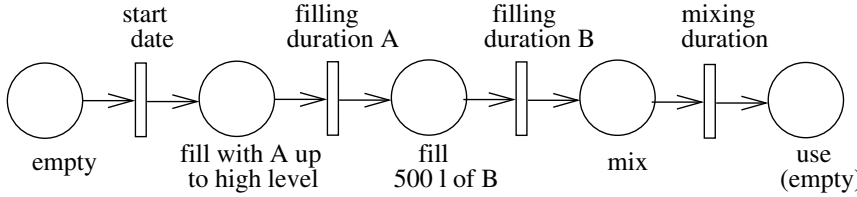


Figure 8.6: Duration not determinable in advance.

However, modeling hybrid systems using temporal Petri nets has certain limitations. For example, consider FIG. 8.6. If the initial level V_o of the product contained in the reactor is not always the same at the beginning of the execution of the recipe, and if the operation to be performed is *fill with product A up to the level* V_{high}, then the duration of the operation is given by

$$\frac{V_{high} - V_0}{q_{i_A}}. \tag{8.10}$$

This duration depends on the value of the continuous state component V at the start of the phase corresponding to the place. Thus, the information contained in the model is no longer sufficient to simulate the system's behavior.

This problem can be addressed using a high-level Petri net (Chapter 5), where a certain number of attributes can be assigned to tokens. Continuous state variables can thus be associated with certain tokens. For example, the variable V can be associated with the token that moves from place to place as an attribute. At the beginning of the operation *fill with 1000 liters of product A*, this attribute will contain the value V_o, which allows the duration to be calculated using formula 8.10. The hybrid model is thus obtained by representing phases as places (discrete state variables) and continuous state variables as attributes associated with tokens.

8.4 Notes

Basic books on fuzzy set theory and possibility theory: DUBOIS & PRADE (1988), KLIR (1988) and BOUCHON-MEUNIER (1994), as well as the articles ZADEH (1977, 1978).

Linear logic is a recent theory developed by GIRARD (1987,1990). For details on sequent calculus see FITTING (1990). For more details on Petri nets and linear logic, see PRADIN (1993).

Modeling hybrid systems using a single model is also recent. Some works using Petri nets: DAUBAS (1994), ANDREU (1994,1995).

Appendix A

Graphs

Graph theory is an old subject with modern applications. The first result was given by Euler in 1736 with the solution to the problem of the city of Koenigsberg: how to cross the city's seven bridges, passing over each one only once.

Until 1946, graph theory was confined to Mathematics. The most important contributions were made by mathematicians such as Kirchhoff, Hamilton, Sylvester, Kempe, Lucas, Petersen, and Tarry in the 19th century, and Poincare, Sainte-Lague, Kuratowski, Hall, Polya, Konig, Whitney, and Tutte in the first half of the 20th century.

Operations research, which emerged from military research related to World War II, spurred the development of graph theory as a model for concrete systems. Among the prominent experts of this new orientation were Konig, Dantzig, Roy, Faure, and Kaufmann. Another significant leap occurred in the 1960s with the development of Information and Communication Sciences, with contributions from computer scientists such as Dijkstra, Knuth, Wirth, Sakharovitch, Gondran, and others (both in the development of algorithms and in problem modeling).

In the study of graphs, it is important to know the algorithms adapted to the main types of graph problems: finding the shortest path, traversing each edge of the graph exactly once, etc.

A.1 Formal Definitions and Notation

Definition A.1.1 (Graph)**.** *A graph or undirected graph $G = (V, E)$ consists of a set V of vertices (or nodes) and a set E of edges such that each edge $e \in E$ is associated with an unordered pair of nodes.*

Definition A.1.2 (Directed Graph)**.** *A directed graph or ordered graph $G = (V, E)$ consists of a set V of vertices (or nodes) and a set E of edges such that each edge $e \in E$ is associated with an ordered pair of nodes.*

Definition A.1.3 (Incident Edge)**.** *An edge (directed or undirected) associated with a pair of nodes v and w is said to be incident on v and w, and the nodes are said to be adjacent.*

Definition A.1.4 (Parallel Edges)**.** *Edges are said to be parallel if they are associated with the same pairs of vertices.*

Definition A.1.5 (Loop). *A loop is an edge incident on a single vertex.*

Definition A.1.6 (Isolated Vertex). *A vertex is considered isolated when there are no incident edges.*

Definition A.1.7 (Pendant Vertex). *A vertex is said to be pendant when there is only one incident edge.*

Definition A.1.8 (Simple Graph). *A graph is simple when it has no loops or parallel edges.*

Definition A.1.9 (Degree). *The degree of a node is the number of edges incident on the node. In the case of a loop, it is counted as two.*

Definition A.1.10 (Multivalued Mapping Γ). *The multivalued mapping Γ is defined by $\Gamma : V \to \mathcal{P}(V)$, where Γ_i is the set of successors of node i, and ${\Gamma_i}^{-1}$ is the set of predecessors of node i.*

A.2 Connectivity

Paths and Circuits

Paths and circuits are defined in a directed graph.

Definition A.2.1 (Path). *Consider v_o and v_n nodes in a graph. A path from v_o to v_n of length n is an alternating sequence of $n+1$ nodes and n edges starting at v_o and ending at v_n.*

Definition A.2.2 (Elementary Path). *An elementary path from v to w is a path that traverses each node exactly once.*

Definition A.2.3 (Circuit). *A circuit is a non-null length path from v to v that traverses each edge exactly once.*

Definition A.2.4 (Elementary Circuit). *An elementary circuit is a cycle from v to v that traverses each node exactly once (except $v_o = v_n = v$).*

Chains and cycles

Chains and cycles are defined for undirected graphs; chains correspond to paths and cycles correspond to circuits for directed graphs in the definitions above.

Definition A.2.5 (Transitive Closure). *The transitive closure of the application Γ is the application $\hat{\Gamma}$ defined by*

$$\hat{\Gamma}_i = i \cup \Gamma_i \cup {\Gamma_i}^2 \ldots \cup {\Gamma_i}^{n-1}$$

${\Gamma_i}^k$ is the set of nodes that can be reached from node i by paths having exactly k edges. $\hat{\Gamma}_i$ is the set of successors *of i. Similarly, $\hat{{\Gamma_i}^{-1}}$ represents the set of* predecessors *of i.*

Definition A.2.6 (Connected Graph). *A graph is connected if for any pair of nodes v and w in G, there exists a chain from v to w.*

For example, it is possible to construct a graph representing the Brazilian states, with an edge connecting the states that share a common border.

A connected graph is unique, while a disconnected graph has several *pieces*, called *subgraphs* or *components*.

Definition A.2.7 (Subgraph). *$G' = (V', E')$ is a subgraph of $G = (V, E)$ if: a) $V' \subseteq V$ and $E' \subseteq E$; b) $\forall e' \in E'$, if e' is incident on v' and w', then v' and $w' \in V'$ (restriction b) ensures that G' is a graph).*

Definition A.2.8 (Component). *Given a graph G and a node v of G, the subgraph G' of G formed by all arcs and nodes of G contained in the chains starting at v is called the component of G containing v.*

A graph is connected if it has only one component. The graph in FIG.A.1.a (JOHNSONBAUGH (1993)) is connected, since for any pair of vertices v and w, there is a path from v to w. However, the one in FIG.A.1.b is not connected; for instance, there is no path from v_2 to v_5.

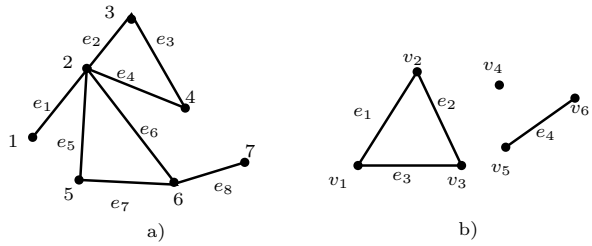


Figure A.1: a) Connected Graph; b) Not connected graph.

Definition A.2.9 (Strongly Connected Graph). *A directed graph G is strongly connected if for any pair of nodes v and w in G, there exists a path from v to w and a path from w to v, $\forall w, v \in V$.*

Definition A.2.10 (Weakly Connected Graph). *A directed graph is weakly connected if for any pair of nodes v and w in the graph underlying graph, there exists a chain from v to w.*

In FIG. A.2, graph G is strongly connected (there is a path between any pair of vertices). G is also weakly connected. On the other hand, graph H is not strongly connected, as there is no path between a and b; however, it is weakly connected, as there exists a chain between any pair of vertices in the underlying graph.

Definition A.2.11 (Eulerian Cycle). *A cycle in G that includes all the arcs and all the nodes is an Eulerian cycle.*

If G consists of a single node v, with no arcs, the path (v) is called an Eulerian cycle in G.

Theorem A.2.1. *If a graph G has an Eulerian cycle, then G is connected and all nodes have even degree.*

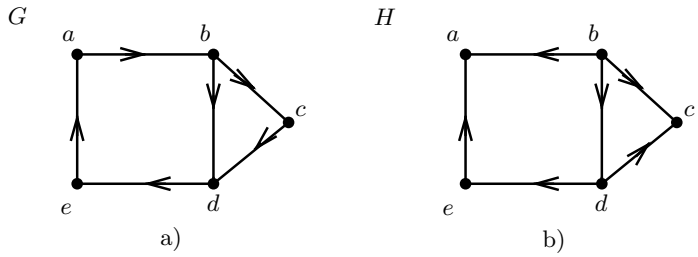


Figure A.2: a) Strongly connected graph; b) Weakly connected graph.

If the node has an odd number of incident vertices, it is impossible to *leave* the node without passing through the arc twice.

Theorem A.2.2. *If G is a connected graph and all the nodes have an even degree, then G has an Eulerian cycle.*

Theorem A.2.3 (Handshaking Lemma)**.** *If G is an undirected graph with m arcs and n vertices, then the sum of the degrees of all nodes, $\sum_{i=1}^{n} \delta(v_i) = 2m$, is even.*

Corollary A.2.1. *In any graph, there is an even number of vertices with odd degree.*

Theorem A.2.4. *A graph G has a path without repeated arcs from v to w ($v \neq w$) containing all arcs and nodes if and only if it is connected and v and w are the only nodes with odd degree.*

Theorem A.2.5. *If G contains a cycle from v to v, G contains an elementary cycle from v to v.*

Algorithm for constructing Eulerian cycles

procedure *Euler* (G: connected graph, $\forall v \in V, degree(v) = even$)
    cycle:= cycle in G starting from any node
    $H := G$ with the arcs of this cycle removed
    while H has arcs
        begin
            subcycle:= cycle in H starting from a node in H that
                    is also a node in *cycle*
            $H := H$ with arcs from *subcycle* and isolated nodes removed
            cycle:= *cycle* with *subcycle* inserted at the appropriate node
        end *cycle* is an Eulerian cycle.

A.3 Notes

For more details about graph theory see JOHNSONBAUGH (1993), GONDRAM & MINOUX (1985) and ROSEN (1991).

Appendix B

Calculation of Conservative and Repetitive Components

B.1 Principle of Calculating a Basis

The method will be explained for finding the conservative components. It is only necessary to transpose the matrix C (swap places and transitions) to find the repetitive stationary components.

We seek the solutions of the equation:

$$f^T C = 0$$

which can also be written as

$$C^T f = 0.$$

The number of variables in this equation corresponds to the number of places in the Petri net (n components of f or n rows of C), and the number of equations corresponds to the number of transitions (m columns of C).

This matrix equation is equivalent to the system of equations:

$$\left.\begin{array}{ccccccccc} c_{11}f_1 & + & ... & + & c_{n1}f_n & = & 0 \\ ... & + & ... & + & ... & = & 0 \\ c_{1m}f_1 & + & ... & + & c_{nm}f_n & = & 0 \end{array}\right\} \; m \text{ equations.}$$

This is a system of linear equations with a zero right-hand side. Therefore, the set of solutions forms a vector space. The degenerate solution $f^T = 0$ exists but is of no interest. What is important is to find a basis for the solution space.

As will be seen next, if r is the rank of C and n is the number of places, the dimension of this vector space will be:

$$dim_p = n - r$$

and the dimension of the repetitive stationary components

$$dim_t = m - r$$

where m is the number of transitions.

There is a systematic method for solving linear equations, called the *Gaussian method*. According to this method, the solutions of a system of linear equations are not altered if the following operations are performed on the rows (or columns):

1. Exchange two rows;
2. Multiply a row by a non-zero scalar;
3. Add one row to another row.

Since what is sought is a basis of solutions, the operations will be performed on the rows of C. Variable changes will be made to seek the triangularization of this matrix.

The following combinations of variable changes will be made:

1. the variable f_i is replaced by af'_i with $a \neq 0$;
2. the variable f_i is replaced by $f'_i + f'_j$ (which is the same as replacing row j by the sum of rows i and j without modifying row i);
3. the variables are reordered when necessary.

If C is not square, the goal is to triangularize a submatrix S of dimension r that corresponds to a system of linearly independent equations that has only one solution (in this case, the degenerate solution).

The columns that do not belong to S (submatrix S') will be linear combinations of the columns of S, and the $n - r$ rows of C that do not belong to S will contain only zeros. In effect, they should correspond to variables that do not appear in any equation and whose value remains free. If $n - r = 0$, then only the degenerate solution exists.

The resulting system of equations is then:

$$\begin{array}{rcl} f'^T C' = 0 & \text{with} & f = Ff' \\ & \text{and} & C' = F^T C \end{array}$$

where F is a regular matrix that describes the variable change and C' has the following form:

$$C' = \left[\begin{array}{c|c} S & S' \\ \hline (0) & (0) \end{array}\right]$$

with:

$$S = \begin{bmatrix} s_{11}=1 & s_{12} & s_{13} & ... & s_{1r} \\ 0 & s_{22}=1 & s_{23} & ... & s_{2r} \\ 0 & 0 & s_{33}=1 & ... & s_{3r} \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & s_{rr}=1 \end{bmatrix}.$$

S' is any matrix.

By writing the first r equations of this new system, we get:

$$\begin{array}{rcl} f'_1 & = & 0 \\ s_{12}f'_1 + f'_2 & = & 0 \\ ..... & = & 0 \\ s_{1r}f'_1 + s_{2r}f'_2 + ... + f'_r & = & 0 \end{array}$$

This system has a unique solution concerning the r variables $f'_1, ..., f'_r$, which is the degenerate solution. Thus, the solutions in the form f' are such that the first r components of this vector are always equal to zero. On the other hand, the other $dim_p = n - r$ components can be chosen freely since they have been eliminated from the system of equations (the last $n - r$ rows of C' are zero).

Obtaining a basis for the vector space of solutions in this form is immediate. It is enough to choose the solutions for which one and only one component of f' is non-zero:

$$\left[\begin{array}{c}(0)\\\hline f'_{r+1}=1\\0\\...\\0\end{array}\right], \quad \left[\begin{array}{c}(0)\\\hline 0\\f'_{r+2}=1\\...\\0\end{array}\right], \quad ... \quad \left[\begin{array}{c}(0)\\\hline 0\\0\\...\\f'_n=1\end{array}\right],$$

which is why the dimension of the vector space of solutions is $n - r$.

To write this basis in the original form of the equation

$$f^T C = 0$$

it is sufficient to construct the matrix F. This provides a basis of conservative components.

As previously mentioned, the Gaussian elimination method consists of introducing zeros by combining the rows of the matrix. Consider the case where the first $i - 1$ rows and columns have been triangularized. Now consider row i (a variable) and column i (an equation). The coefficients of this column corresponding to the variables $i + 1$ to n are zeroed by multiplying row i by c_{ki}, row k by c_{ii}, and eliminating row i from row k.

If the coefficient c_{ii} is zero, such an operation is impossible. In this case, permutations among the last $m - i$ columns of C (which are being transformed) must be performed to produce a non-zero c_{ii}. If this is impossible, it means that row i contains only zeros since the first $i - 1$ columns of C have already been processed. The variable f_i has already been eliminated from the system of equations and produces an element of the basis of the vector space of solutions. The last $n - i$ rows of C are permuted to produce a row that does not contain only zeros. If this is impossible, the algorithm terminates; C has been written in the form C'.

Replacing "row k" with "c_{ii} times row k minus c_{ki} times row i" involves the following variable change (allowed for $c_{ii} \neq 0$):

$$\begin{array}{rcl} f_i & = & f'_i - c_{ki} f'_k \\ f_k & = & c_{ii} f'_k \\ f_j & = & f'_j \ \ \forall j \neq i, k. \end{array}$$

Thus, the equation j (column j of C), for example, is written as:

$$c_{1j} f_1 + ... + c_{ij} f_i + ... + c_{kj} f_k + ... + c_{nj} f_n = 0$$

becomes:

$$c_{1j} f'_1 + ... + c_{ij}(f'_i - c_{ki} f'_k) + ... + c_{kj} c_{ii} f'_k + ... + c_{nj} f'_n = 0$$

or:

$$c_{1j}f'_1 + ... + c_{ij}f'_i + ... + (c_{ii}c_{kj} - c_{ki}c_{ij})f'_k + ... + c_{nj}f'_n = 0.$$

If we consider equation i ($j = i$), we find that the variable f'_k is thus eliminated from the equation. Normalizing row i in equation i (so that the coefficient s_{ii} of S is equal to 1) typically involves making the following change:

$$f_i \;=\; \tfrac{1}{c_{ii}} f'_i.$$

In fact, this normalization is unnecessary, as it does not alter the principle of the method and may introduce fractional numbers into the calculations.

The matrix F can also be constructed through successive transformations. We start with an identity matrix and perform the same operations on the columns of F as we do on the rows of C. In the previous case, this consists of replacing column "k" with "c_{ii} times column k minus c_{ki} times column i" in the matrix F. Such a procedure is justified because this variable change is described by:

$$f \;=\; \begin{matrix} 1 \\ i \\ k \\ n \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -c_{ki} & 0 \\ 0 & 0 & c_{ii} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} f'$$

and that the matrix F of the variable change resulting from the chain of elementary variable changes A_1, A_2, ... A_l is equal to:

$$F = A_1 A_2 \ldots A_l.$$

B.2 Example

Consider the Petri net in FIG. 2.23, with $P = \{p_1, p_2, p_3, p_4, p_5\}$. Initially, we have:

$$C_0 \;=\; \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \qquad F_0 \;=\; \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Eliminating the first row relative to the first column is done by multiplying the first row $C(1, .)$ by -1, and replacing $C(2, .)$ with $C(2, .) + C(1, .)$ and $C(3, .)$ with $C(3, .) - C(1, .)$. The first row will no longer be treated and will correspond to the first row of S; it will not be part of the generating basis of the solution vector space. Thus, we obtain:

$$C_1 \;=\; \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \qquad F_1 \;=\; \begin{bmatrix} -1 & 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

If it is impossible to make a 1 appear in a row of zeros, one can proceed by swapping rows 2 and 5, as well as columns b and d. The reordering of columns (the order in which the equations are written) does not alter F. Thus, we have:

$$C_2 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 3 & -3 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad F_2 = \begin{bmatrix} -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

The second row can now be eliminated from the solution space by substituting:

- $C_2(3, .)$ by $C_2(3, .) - 3C_2(2, .)$
- and $C_2(4, .)$ por $C_2(4, .) + C_2(2, .)$.

They are then obtained:

$$C_3 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad F_3 = \begin{bmatrix} -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & -3 & 1 & 0 \end{bmatrix}.$$

The last three rows being zero, the triangularization is complete. The solution space is of dimension 3, and the last three columns of F_3 form a basis of this space. They correspond to the last three components of the solutions in f', i.e., after the change of variables.

The conservative components are:

$$\begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \\ -3 \end{bmatrix}, \qquad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

which provides the following linear invariants for the initial marking indicated in FIG. 2.23:

$$\begin{aligned} -M(1) + M(3) - 3M(5) &= -1 \\ M(1) + M(2) &= 1 \\ M(4) + M(5) &= 1. \end{aligned}$$

B.3 Simplified Algorithm and Search for Positive Solutions

In fact, only the calculation of the columns of the matrix F corresponding to the $n - r$ components of the basis of the solutions f is useful. The other columns of F can be eliminated.

At each step (elimination of a variable), the column of C (which represents the equation that will be placed in the basis of equations) used to eliminate a variable (where all elements of the column are zero except one) is removed. Additionally, the row corresponding to this variable is also eliminated. Since this variable is not part of the solution basis, its corresponding column in F is deleted as well."

Finally, instead of explicitly constructing F, the operations performed on the rows of C are memorized as formal sums. Each column of$F$ is stored as a linear combination of the original columns ($f_i + f_j$ describes a column obtained through the sum of columns i and j of F_0).

On the other hand, positive solutions are generally the most interesting (which is evident for stationary repetitive components). Therefore, it is useful to favor positive line combinations at the moment of elimination.

The simplified algorithm is then obtained as follows:

1. Search for a column that has only one non-zero term. It corresponds to an equation in the basis of equations and is deleted, with the corresponding variable to the non-zero term being eliminated, since the only possible solution is zero, and it cannot be part of the solution basis. Delete the row of C associated with this variable and the corresponding column of F. Repeat step **1** as long as possible, then proceed to **2**.

2. Search for a column having only one non-zero component with a given sign; the others must be zero or of the opposite sign (one positive and all others negative or zero, for example). The variable corresponding to this line is eliminated by making only positive line combinations to cause zeros to appear in the considered column. After this operation, eliminate this column, the row of C corresponding to the eliminated variable, and the associated column of F. If such a column is found, return to **1**; otherwise, proceed to **3**.

3. Search for a column having $i \geq 2$ non-zero positive components and $j \geq 2$ non-zero negative components. With the help of one of the non-zero positive components, nullify $j - 1$ negative components. Return to step **2** to eliminate the variable that corresponds to a single non-nullified negative component. It should be noted that a positive component among i and a negative component among j were chosen. Therefore, there are $i \times j$ ways to proceed. This choice can have consequences on the form of the obtained basis. If no column meeting the above criterion is found, proceed to step **4**.

4. If there is a column having all components of the same sign, proceed with Gaussian elimination for this column (but the basis will not have only positive solutions); then return to **1**. Otherwise, the algorithm ends. Either all columns are zero, or the matrix has no more elements. The non-deleted combinations corresponding to the construction of columns of F directly provide the basis vectors.

Considere novamente o exemplo da FIG. 2.23.

$$C_0 = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix}.$$

Observe the first column of C_0, whose corresponding equation is selected to be part of the basis of equations. To eliminate the variable f_2, simply add the rows corresponding to the variables f_1 and f_3 to the row corresponding to f_2. With these new variables, the equation will have only one non-zero term. The row f_2 is eliminated (f_2 will not be part of the solution basis), as well as the first column of C_0. This results in:

$$C_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -3 & 3 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_1 + f_2 \\ f_2 + f_3 \\ f_4 \\ f_5 \end{bmatrix}.$$

The second column of C_1 is used to eliminate f_4 from the solution basis. This is achieved by adding three times the row"f_4"to the row "$f_2 + f_3$" and once to "f_5". The second column of C_1 is then eliminated, along with the row corresponding to the variable f_4. This results in:

$$C_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} f_1 + f_2 \\ f_2 + f_3 + 3f_4 \\ f_4 + f_5 \end{bmatrix}.$$

Only zeros remain, and the algorithm terminates. The resulting basis corresponds to the sums on the right side of the matrix, describing the manipulations performed on the rows of C and the columns of F. By substituting the symbols f_i with the i-th column of F_0, the basis is obtained:

$$\begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{matrix} \quad \mathbf{f}^1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{f}^2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 3 \\ 0 \end{bmatrix}, \quad \mathbf{f}^3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

For the sake of readability, the vector $\mathbf{f}^i$ of conservatives components can be written as a set of places $f^i = \{p_j \mid f^i_j \neq 0\}$. The invariants f^1, f^2, and f^3 can be expressed as:

$$f^1 = \{p_1, p_2\}, \quad f^2 = \{p_2, p_3, p_4^3\}, \quad f^3 = \{p_4, p_5\}.$$

Alternatively, the set notation can be omitted:

$$f^1 = p_1\ p_2, \quad f^2 = p_2\ p_3\ p_4^3, \quad f^3 = p_4\ p_5.$$

The same notation applies for repetitive components.

Appendix C

Shopfloor modeled by Petri nets

The goal of this appendix is to model, from a textual specification, the dynamic behavior of a system using ordinary Petri nets (PN). You can use the Tina[1] toolbox to assist you with edition, simulation and analysis. You must pay attention to the different processes and to their interactions: sequence, parallelism (or independence) and conflict (or choice).

Simulation can be used on partial Petri nets during the first steps of modeling in order to check if the interactions between the processes correspond to the specification. Remember that simulation is not exhaustive, so, formal analysis (by invariants or marking graph and model checking) is necessary. Simulation can be used again to verify, for example a transition invariant, or to simulate a sequence leading to a deadlock.

In order to verify if the model is consistent with the specifications, it is necessary to analyse it by using the reachable marking graph (or recovery tree is the graph is unbouded) and by a structural analysis. It is necessary to interpret the obtained results: are the marking coherent? Do the firing sequences correspond to the expected behavior? Are the place invariants coherent?

The system considered in this appendix is the system given as the Exercise 4 in Chapter 1 with some new operating assumptions. Let us remember the specifications. It is composed of a robot R, a lathe T, and a laser micrometer M (FIG. C.1), interconnected by a communication network, which operates as follows:

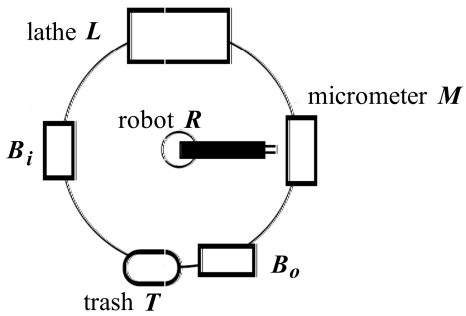


Figure C.1: Shopfloor.

[1]TINA has been developed at LAAS, see more information here: https://projects.laas.fr/tina/index.php

- The parts are deposited in the *input buffer* B_i;
- The robot picks up one part from B_i and places it in the lathe L (if L is free). A program must be loaded to machine (i.e. turn) the part;
- After turning (i.e. machining), the robot picks up the part from the lathe L and places it in the laser micrometer M (if it is free) for being mesured;
- After measurement, the robot R places the part in the *output buffer* B_o if it meets the standards, or in the trash T if it does not. The capacity of the trash and B_o are infinite.
- The robot can only do one operation at a time; the lathe L and the micrometer M can only work on a part at a time and they have no buffer.
- The operation duration on L is longer than that on M.

It will be assumed that the capacity of B_i is two parts.
In addition the following new operating assumptions are considered:

- the system operates correctly: if the robot takes the part, it will not loose it
- the input buffer B_i has a sensor which detects the presence of a part;

In section 1.6 there are some hints for the construction of a Petri net model. They will be presented here in more detail.

There are two main ways for modeling a system using Petri nets:

- Direct Approach: Directly build a *overall* model with a unique Petri net, considering all the elements of the system and their interactions. Chapters 1 and 2 have presented some (small) models using this approach.
- Modular Approach: For more complex systems it is generally necessary either to use a top-down approah, as the stepwise refinement seen in Section 3.7.1, or use a bottom-up approach, by constructing sub-system models that are merged in a overall one by using synchronous or asynchronous composition, as presented in Section 3.7.2.

In all cases, it is necessary to verify and validate the models.

C.1 Overall Modelling

In this appendix, we will use an ordinary Petri Net to represent the system and the Tina toolbox to simulate and analyse it. It is also possible to analyse the Petri net using the algorithms seen in Section 3.1 (state space analysis) and Section 3.2 (structural analysis).

C.1.1 Model construction

C.1.1.1 Determining activities and events

How to describe a discrete event system in terms of activities and events? It's not always easy to start from scratch. Events are typically named with verbs in the imperative form, signifying a specific moment of change or observation, and are associated with transitions in the Petri net. Activities, on the other hand, are named with verbal noun (ending in *ing*) or past tense verbs (ending in *ed*), indicating ongoing processes, actions or states. They are associated with places in the Petri net.

The first thing to do is to create a table, as Table C.1, where each activity is expressed as a function of some events and/or resource utilizations. In other words, activities and resource states indicate the preconditions to enable events and events produce new activities and/or new resource states (the postconditions).

A transition in a Petri net, as well as an operator (or action) in planning theory, is defined on the basis of preconditions and postconditions. Unlike the operator in planning theory, where, in addition to the effect of an action a, the effect$^-(a)$, must be indicated, the firing of a transition in a Petri net removes tokens from its input places.

Activity	**Input Event**	**Output Event**
L is turning	take a part from B_i and put it on L	take the part from L and put it on M
...	...	...

Table C.1: Table of activities.

It is also possible to create a table, as Table C.2, where for each event (associated with a transition), the preconditions needed for enabling the event, and the postconditions obtained after the occurrence of this event are given.

Event	**Precondition**	**Postcondition**
e2: take a part from B_i and put it on L	R is free a part is in B_i	L is turning
...	...	...

Table C.2: Table of events.

While filling one of these tables, or even both of them, some inconsistencies can be found. For example, in Table C.2, it is necessary to check if the lathe L is free, and *preallocate* it, before taking the part from B_i. Indeed, if the lathe is occupied, and if the robot take the part in B_i, there is a deadlock since the robot cannot remove the part already present in the lathe as it has a new part. It is necessary to add the precondition L *is free*.

Another problem is the level of detail. In Table C.2: it is assumed that the duration of the activity of the robot R when it moves the part from B_i to L is negligible compared to that of the operation on L. This duration can be taken into account by breaking down the event in Table C.2 into two events, creating two new activities as indicated in the Table C.3. However the size of the resulting Petri net will be larger.

In parallel with filling out the tables, partial Petri nets can be drawn or edited. The graphical version can help detect inconsistances. Tables C.1, C.2 (with precondition "L

Event	Precondition	Postcondition
take a part from B_i	R and L are free a part is in B_i	a part is moving from B_i to L
put the part on L	a part is moving from B_i to L	L is turning

Table C.3: Considering the moving operation as an activity.

is free" added), and C.3 allow for generating the Petri nets in FIG. C.2, C.3, and C.4, respectively.

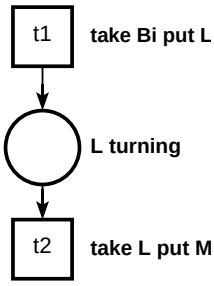


Figure C.2: Table C.1.

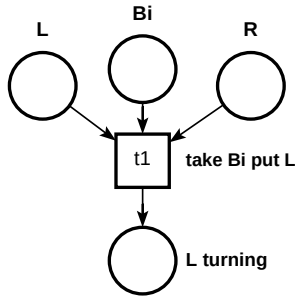


Figure C.3: Table C.2.

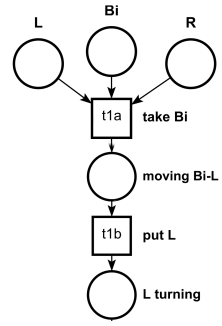


Figure C.4: Table C.3.

Proceeding with the brainstorming for describing the activities of the system, Table C.4 was filled. The ressources, described in Table C.5, will also be associated with places in the Petri net. One could also add to this table the input event that allows acquiring the resource and the output event that allows releasing the resource. Other events occurring on the shopfloor are described in Table C.6. Note that the names of places and transitions were associated with the activities and events, respectively.

Activity	Input Event	Output Event
part in B_i	...	...
M measuring	...	...
part in B_o	...	...
part in trash T	...	...

Table C.4: Other activities.

Resources
robot R
lathe L
micrometer M
input buffer B_i

Table C.5: Ressources.

Event	Precondition	Postcondition
e1: put part in B_i	...	...
e2: take part from B_i to L	...	...
e3: take part from L to M	...	...
e4: take part from M to B_o or T	...	...
e5: put part on T	...	...
e6: put part on B_o	...	...

Table C.6: Other events occurring in the shopfloor.

C.1.1.2 Constructing the Petri net

Once the activities and events are listed, the Petri can then be drawn. At this point, new elements may be added, and other elements and activities may be broken down or deleted. It is necessary to verify that all listed resources are correctly identified and to determine if there are any additional resources in the system. The list of activities and events may be updated if necessary.

The Petri net in FIG. C.5 shows a possible model of the shopfloor. In this model, you can find the activities and events described in Tables C.1 and C.2. Some changes were made.

- After event e2 the robot R is free again
- Event e3 "take part from L to M" was broken down into two events: e31 (remove the part from the lathe L), and e32 (put the part in the micrometer M).
- Event e4 "take part from M to B_o or T" has been broken down into three events: e41 (remove part from M), e5 (put part in T) and e6 (put part in B_o).
- Transitions e7 and e8 in FIG. C.5 enable the introduction of a new part in the shopfloor after the execution of the final operation on a part. Both allow representing event e1 in Table C.6, which is removed from the list. Note that the firing of e7 and e8 also allows representing the states "part in T" (place p9) and "part in B_o" (place p10), respectively.

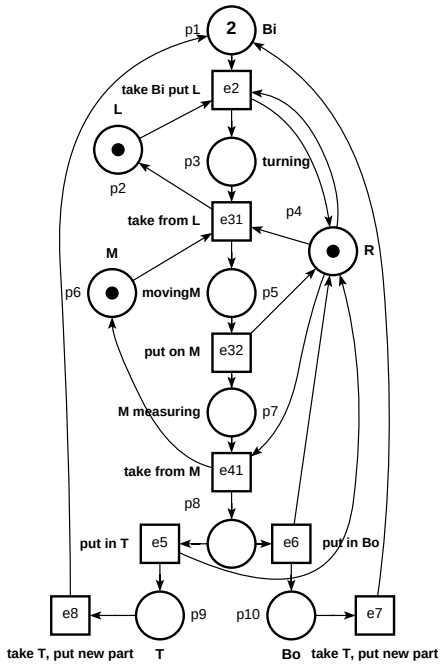
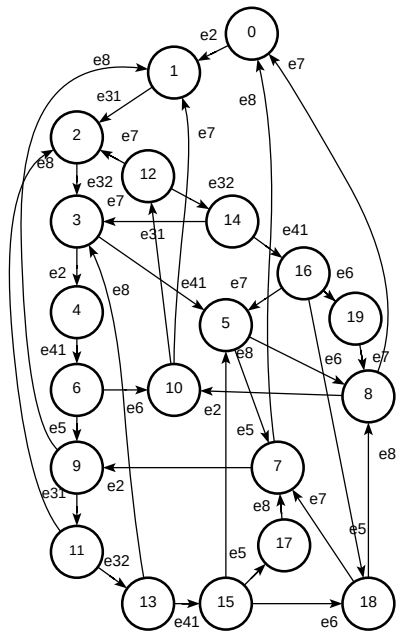


Figure C.5: overall Petri net.




```
0 : Bi*2 L M R
1 : Bi M R p3
2 : Bi L p5
3 : Bi L R p7
4 : R p3 p7
5 : Bi L M p8
6 : M p3 p8
7 : Bi L M R T
8 : Bi Bo L M R
9 : M R T p3
10 : Bo M R p3
11 : L T p5
12 : Bo L p5
13 : L R T p7
14 : Bo L R p7
15 : L M T p8
16 : Bo L M p8
17 : L M R T*2
18 : Bo L M R T
19 : Bo*2 L M R
```

Figure C.6: Marking graph and markings.

Others Petri net model can be designed. If the information given by p9 and p10 is not useful for the user, place p9 can be removed and transitions e5 and e8 can be merged

in a unique transition e58, "remove part from T and put a new part in B_i". The same can be done with p10 and transitions e6 and e7.

Another model can be obtained if the user wants to introduce a new part as soon as a part is removed from the B_i. In this case, just remove places p9 and p10, transitions e7 and e8, and create an arc (e2, B_i). Several solutions are possible depending on the chosen level of abstraction.

Whatever the chosen model, it must be verified and validated, as extensively discussed in Section 3.6. This is the next step in the modeling of the shopfllor and will be done in the following.

C.1.2 Formal verification

We will use the analysis based on the reachable marking graph (or coverability graph if the marking graph is unbounded) and structural analysis seen in Chapter 3. The results of the analysis of the overall model in FIG. C.5 must then be interpreted by the user: Are markings consistent? Do the firing sequences match the expected behavior? Are place invariants consistent? Do a repetitive component truly correspond to a firing sequence?

For the sake of readability some places in FIG. C.5 are labeled : place p1 is Bi, p2 is L, p4 is R, p6 is M, p9 is T and p10 is Bo.

C.1.2.1 Analysis by marking graph

The marking graph[2]of the Petri net of FIG. C.5 is shown in FIG. C.6. The Petri net is bounded, live and reversible.

Let us check some markings and verify that they are consistent:

- marking `19: Bo*2 L M R`: There are two parts in the output buffer; the lathe, the robot and the micrometer are available;
- marking `17: L M R T*2`: There are two parts in the trash; the lathe, the robot, and the micrometer are available;
- marking `15: L M T p8` : The robot has the part taken from the micrometer (p8); the lathe and the micrometer are available; there is a part in the trash;
- marking `4: R p3 p7` : There is one part turning and another part in the micrometer; the robot is available;
- In all 20 markings, place R has one token or is not marked, indicating that the robot is available or occupied, respectively. The same applies to place L (lathe) and M (micrometer).

These markings are consistent.

[2]The marking graph (textual and graphical versions) were generated using Tina as explained in Appendix E.

C.1.2.2 Structural analysis

The conservative component are calculated by applying Equation 2.28, $\mathbf{f}^T C = 0$ with $\mathbf{f} > 0$. Instead of writing the vector $\mathbf{f}_i$ with elements $\mathbf{f}_i(p) \in I\!N$, it will be written with the places p that have $\mathbf{f}_i(p) = 1$. This notation is used for the sake of readability, as introduced on page 2.5.1.

Using this notation, the conservative components for the Petri nets in FIG. C.5 are as follows: $f_1 = \{L, p3\}$, $f_2 = \{M, p5, p7\}$, $f_3 = \{R, p5, p8\}$, and $f_4 = \{Bi, Bo, T, p3, p5, p7, p8\}$.

We verify again that the net is bounded (without using the marking graph): each net place belongs to at least one conservative component. We can also calculate the bound of each place and, more important, interpret the corresponding states by calculating the place invariants according to Equation 2.29, $\mathbf{f}^T M = \mathbf{f}^T M_0$.

For the sake of readability, place invariants will be written using the name of places p directly instead of $M(p)$ as in equation 2.29. Using this notation, for initial marking M_0 =`Bi*2 L M R`, the place invariants are as follows:

1. L + p3 = 1:
   This place invariant describes the behavior of the lathe: either it is available (L=1) *or* it is turning (p3=1);

2. M + p5 + p7 = 1:
   Describes the behavior of the micrometer that may be in one of the following states: available (M=1), measuring a part (p7=1), *or* assigned to a part (p5=1, robot is already moving a part to the micrometer);

3. R + p5 + p8 = 1:
   Describes the behavior of the robot, which may be in one of the following states: available (R=1), moving a part to the micrometer (p5=1), *or* moving a part from the micrometer (p8=1);

4. Bi + Bo + T + p3 + p5 + p7 + p8 = 2
   Describes the behavior of the parts: no more than two parts are present on the shopfloor since $M(p) \leq 2, \forall p \in \mathbf{f}_4$. Several reachable markings (shown in FIG. C.6) illustrate this constraint : marking 7 (one part in B_i and one part in T, all resources free), marking 8 (one part in B_i and one part in B_o, all resources free), etc.

Place invariants 1 to 3 show that resources have been well modeled. They cannot be simultaneously in two different states. Invariant 4, does not prove this. It does not exclude that two parts might be in the micrometer (p7=2). It is necessary to analyse another invariant containing the same place. For example, invariant 2 indicates that (p7=2) is impossible because markings are positive or null integers. This illustates the fact that all the invariants have to be considered simultaneously.

Section 3.7 presented two approaches for the modeling of complex and/or large systems. The following section will illustrate the synchronous composition in the case of the shopfloor.

C.2 Modular modeling by synchronous composition

This section will apply the synchronous composition of Petri net modules. As seen in Section 3.7.2, we first need to decompose the system into modules. We start from a detailed Petri net for each module, which must be verified and validated. Then, a overall Petri model is obtained by merging the transitions of the individual Petri nets. This resulting Petri net must also be verified and validated, as the properties of the modules may not be preserved.

The steps are as follows:

1. Using an object-based decomposition, the objects or modules of the shopfloor are identified as the input buffer, robot, lathe, and micrometer. The Petri net models of these modules, R_{B_i}, R_R, R_L, and R_M, respectively, are represented in FIG. C.7. Since the trash and output buffer are infinite, they cannot be modeled by a Petri net because they would be unbounded.

   After verification by marking enumeration, all Petri nets models are founded to be bounded, live and reversible. The input buffer Petri net is 2-bounded, while the three others are safe (binary) Petri nets.

Table C.7: Transitions to be merged and its associated events

Transition	Associated event	Table C.6
t8	take part from B_i and put it on L; add a new part in B_i	e1+e2
t9	take part from L to M; allocates the micrometer; move to M	e31
t10	put part on M (L and R are released)	e32
t11	take from M (and acquire the measuring result)	e4
t12	put part on T	e5
t13	put part on B_o	e6

2. The synchronous composition is achieved by merging the transitions associated with the same event. The approach involves considering an event in the system and identifying the corresponding transition in each sub-model. This has already been done in the Petri net models of FIG. C.7 where the same transition names are used for the same events. These transitions are listed in Table C.7.[3]

   Following the definition of synchronous composition introduced in Section 3.7.2.1, we can merge the nets two by two:

   - Overall net R_1 (R_{B_i} and R_L): $T_c = \{t8\}$, $T_{B_i i} = \phi$, $T_{Li} = \{t7, t9\}$;
   - Overall net R_2 (R_1 and R_M): $T_c = \{t9\}$, $T_{1i} = \{t7, t8\}$, $T_{Mi} = \{t10, q7, t11\}$;
   - (Final) overall net R (R_2 and R_R): $T_c = \{t8, t9, t10, t11\}$, $T_{2i} = \{t7, q7\}$, $T_{Ri} = \{t12, t13\}$.

3. The resulting (overall) Petri net R is given in FIG. C.8. Transitions t7 (*start L operation*), q7 (*start M operation*), t11 (*put on T*) and t12 (*put on B_o*) are *internal* transitions. After verification by marking enumeration, the overall Petri net model

[3] Tina allows for the synchronous composition of Petri nets, see Appendix E.3.

is founded to be bounded, live and reversible. All places are safe (binary) places, except B_i which is 2-bounded.

Let us consider places p11 in FIG. C.7.c and p5 in FIG. C.7.d. They appear as two identical places in the overall model in FIG. C.8 (see Section 3.3.2.2). One of them can removed, e.g., p11. The predicate of p5 can be modified to *M is assigned & robot is moving a part to it.*

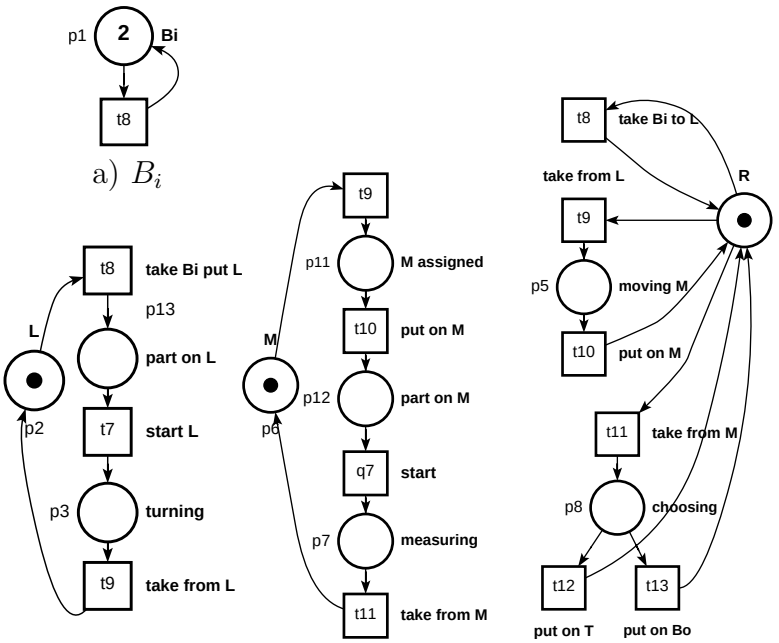


Figure C.7: Individual Petri nets.

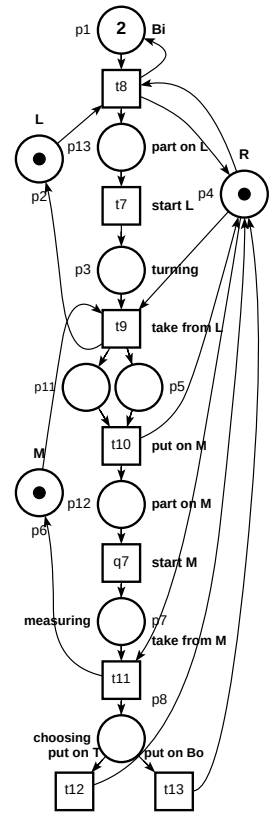


Figure C.8: overall PN.

If you compare the overall Petri net in FIG. C.5 and the one in FIG. C.8, you can notice some differences. The model in FIG. C.8 is more detailed, but also the part feeding strategy is different. A new part is introduced into B_i as soon as it is taken by the robot R, whereas in the model in FIG. C.5 it is introduced when a part is put into B_o or T. In order to obtain this second behavior, the Petri net in FIG. C.7.a should be modified by adding a place *"a part is been turned and measured"* and two transitions t_{12} and t_{13}.

C.3 Exercises

1. Consider the Petri net model in fig. C.9. The validation using the marking graph indicates that the model is not live. Why this model has a deadlock? Compare transitions e31 and e32 with the ones in fig. C.5. What is difference? Hint: Simulate the scenario where a part is taken from the input buffer, put in the lathe and start turning, then a new part is taken from the input buffer.

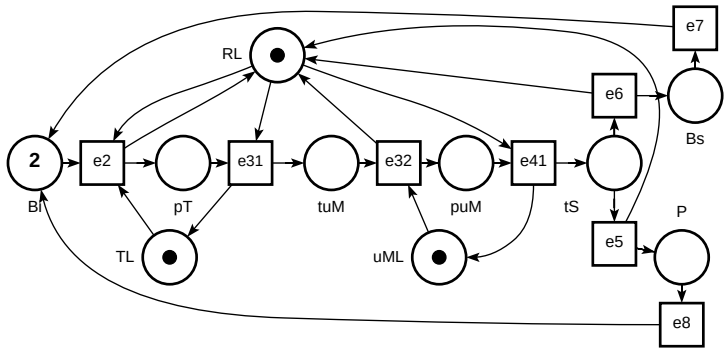


Figure C.9: A bad model of the shopfloor.

Appendix D

Composition of Petri Nets Models

The following examples demonstrate the modeling of different itineraries in a transportation system, similar to the one discussed in Example 1.1. Since each itinerary is physically composed of sections, its model can be obtained through the composition of the Petri net models of these sections.

Example D.1 describes the model and analysis of a single section. Example D.2 explains a synchronous composition of two sections in sequence, while Example D.4 details the asynchronous composition of two intersecting sections.

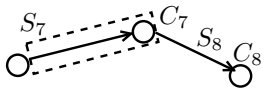


Figure D.1: Two sections in sequence.

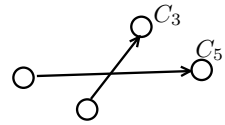


Figure D.2: Two intersecting sections.

D.1 Synchronous composition

Example D.1 Modeling one section
FIG D.3 illustrates the Petri net model R_s for a single vehicle per section. Labels have been added for clarity: M represents "vehicle moving in the section", C for "vehicle stopped at the contact", and $Sfree$ for "section is free". Transitions t_1 and t_3 correspond to actions "vehicle enters the section" and "vehicle leaves the section", respectively, while transition t_2 represents "vehicle stops at the contact".

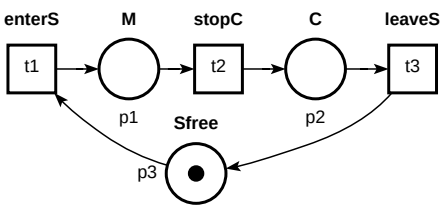


Figure D.3: Model R_s (one section).

$$C_s = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 \end{matrix} \\ \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}$$

$\mathbf{f_s} = [1 \;\; 1 \;\; 1]^T$ or `p-semi-flow:` $\mathtt{p_1p_2p_3}$
$\mathbf{s_s} = [1 \;\; 1 \;\; 1]^T$ or `t-semi-flow:` $\mathtt{t_1t_2t_3}$.

Figure D.4: Structural analysis of R_s.

Matrix C_s of the Petri net R_s shown in FIG D.3 is depicted in FIG D.4. This figure also presents two notations for the conservative and repetitive components: the vector

notation $\mathbf{f_s}$ and $\mathbf{s_s}$, and the semi-flow notation given by Tina when using the structural analysis calculation.

From the conservative component $\mathbf{f_s} = [1 \quad 1 \quad 1]^T$, it can be proven that the net is bounded. As all transitions appear in the repetitive component $\mathbf{s_s} = [1 \quad 1 \quad 1]^T$, no conclusion can yet be drawn about liveness. However this Petri net is a state machine (see Section 3.5.2) and there is a token in its repetitive component which is strongly connected. Therefore the net is live and reversible.

The sequence $s_s = t_1 t_2 t_3$ is a realizable t-invariant from the initial marking $M_s = p_3$. This means that the arrival of a vehicule in the section (t_1), the fact that it is moving (M), its arrival at the contact (t_2), its stay at the contact (C) and its exit from the section is the repetitive behavior of the system.

The following place invariant I_s (using the place labels) is derived for the initial marking $M_s = p_3$:

$I_s : M(M) + M(C) + M(Sfree) = 1$.

This p-invariant shows that the vehicule is either moving in the section (M) stopped at the contact (C), *or* the section is free ($Sfree$). ◇

Example D.2 Synchronous composition

Consider an itinerary with two sections, S_7 and S_8, in sequence, as in FIG. D.1. Each section can be modeled by the Petri net R_s in FIG. D.3. The Petri net model of this itinerary can be obtained by the synchronous composition of the two Petri nets, R_{s7} and R_{s8}, both based on the model R_s described in Example D.1. The resulting model is presented in FIG. D.5, where transitions labeled by "leaveS7" (t_3 in R_{s7}) and "enterS8" (t_3 in R_{s8}) are merged into a single transition t_3, labeled "leaveS7enterS8", as they correspond to the same event (to leave S_7 and to enter the following section, S_8).

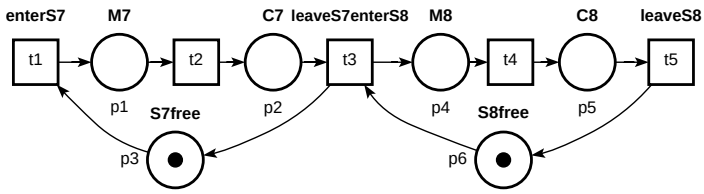


Figure D.5: Model R (two sections).

$$C = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix} \end{array}$$

Figure D.6: Matrix C of R.

Following the definitions in Section 3.7.2.1, the composed model R has $P = P_{s_7} \cup P_{s_8} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, and $T_s = T_{s_7 i} \cup T_{s_8 i} \cup T_c$, with $T_{s_7 i} = \{t_1, t_2\}$, $T_{s_8 i} = \{t_4, t_5\}$, and $T_c = \{t_3\}$. Matrix C of the Petri net R, shown in FIG D.5, is depicted in FIG D.6.

The Petri net R has two conservative components, $\mathbf{f_1}^T = [\ \mathbf{f}_{s_7}^T \ 0 \ 0 \ 0\ ]$, and $\mathbf{f_2}^T = [\ 0\ 0\ 0\ \mathbf{f}_{s_8}^T\ ]$, or equivalently, $\mathbf{f_1}^T = [\ 1\ 1\ 1\ 0\ 0\ 0\ ]$, $\mathbf{f_2}^T = [\ 0\ 0\ 0\ 1\ 1\ 1\ ]$. No new conservative component emerges, but the conservative components f_{s_7} and f_{s_8} are preserved indicating that the overall net remains bounded.

The overall net R has a single repetitive component, $\mathbf{s}^T = [\ 1\ 1\ 1\ 1\ 1\ ]$. It is a new one that has emerged. No conclusions can yet be drawn about liveness. However, this net is an event graph (Section 3.5.3) and there is one token in each elementary circuit;

therefore, the net is live. Notice that there are no repetitive components in the subnets involving only the internal transitions of T_{s7i} or T_{s8i}, so none are preserved.

Considering the marked Petri net with marking $M^T = [M_{s7}^T \;\; M_{s8}^T]$, or $M = p_3p_6$, the sequence $s = t_1t_2t_3t_4t_5t_6$ is a realizable t-invariant. This means that the repetitive behavior of the system corresponds to a vehicle crossing successively the two sections s_7 and s_8.

For the initial marking, the p-invariants corresponding to conservative components f_1 and f_2 (using the place labels) are:

$I_1 : M(M7)+M(C7)+M(S7free) = 1, \qquad I_2 : M(M8)+M(C8)+M(S8free) = 1.$

We can see that the internal p-invariants are preserved in the overall model which involves that there is at most one vehicle in each section. ◇

D.2 Asynchronous composition

Example D.3 Modeling one cell

Consider an itinerary with two intersecting sections, S_3 and S_5, as shown in FIG. D.2. In this case, the vehicle must check if the crossing cell is free, in addition to verifying if the section is available. FIG D.7 depicts a model that takes this additional resource into account (place p_4, labeled $CellFree$).

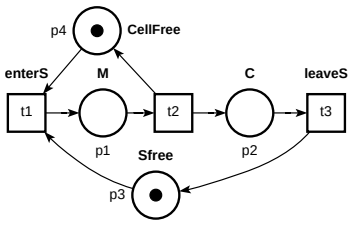


Figure D.7: Model R_c (one cell).

$$C_c = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 \end{matrix} \\ \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{matrix}$$

$\mathbf{f}_{c1} = [1 \;\; 1 \;\; 1 \;\; 0]^T, \;\; \mathbf{f}_{c2} = [1 \;\; 0 \;\; 0 \;\; 1]^T$
$\mathbf{s}_c \;\; = [1 \;\; 1 \;\; 1]^T.$

Figure D.8: Structural analysis of R_c.

The incidence matrix C_s of the Petri net R_s, shown in FIG D.7, is depicted in FIG D.8. This figure also presents the conservative and repetitive components using the vector notation. It is interesting to note that place p_4 ($CellFree$) is implicit (see Section 3.3.2.1) with respect to the places p_2 (C) and p_3 ($SFree$). In consequence, the petri net of FIG. D.7 has the same properties (bounded, live and reversible) as that of FIG. D.3.

For the marking $M_c = p_3p_4$ the (ordered) sequence $s_c = t_1t_2t_3$ is a realizable t-invariant. This means that the repetitive behavior of the system is preserved. The place invariants for this Petri net are as follows:

$I_{c_1} : M(M) + M(C) + M(Sfree) = 1$, which has the same meaning than I_s in Example D.1.

$I_{c_2} : M(M) + M(CellFree) = 1$: meaning the cell is either free or there is a vehicle moving in it. ◇

Example D.4 Asynchronous composition

The Petri net model for a cell with two sections that intersect can be obtained with the asynchronous composition of two Petri nets as in FIG D.7, by merging places $CellFree$. The resulting model is presented in FIG. D.9.

Following the definitions in Section 3.7.2.2, the composed model R' has $T' = T_{s_3} \cup T_{s_5} = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, and $P' = P_{s_3 i} \cup P_{s_5 i} \cup P_c$, with $P_c = \{p_4\}$, $P_{s_3 i} = \{p_1, p_2, p_3\}$, $P_{s_5 i} = \{p_5, p_6, p_7\}$.

The incidence matrix C' of the overall Petri net R', shown in FIG D.9, as well as its conservative and repetitive components (using the set notation introduced on page 61 and page 64) are depicted in FIG D.10.

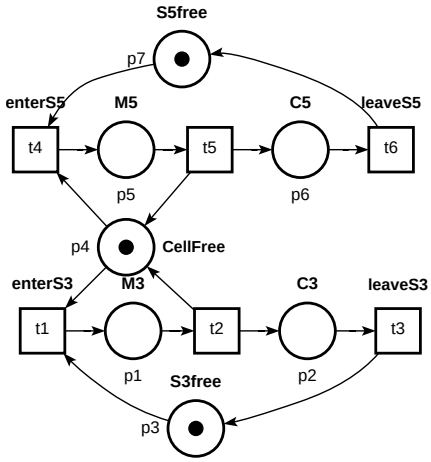


Figure D.9: Two crossing sections.

$$C' = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array} \begin{array}{c} \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{array} \\ \left[\begin{array}{cccccc} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{array}\right] \end{array}$$

$\mathbf{f'}_1 = \{p_1, p_2, p_3\}$, $\mathbf{f'}_2 = \{p_5, p_6, p_7\}$,
$\mathbf{f'}_3 = \{p_1, p_4, p_5\}$
$\mathbf{s'}_1 = \{t_1, t_2, t_3\}$, $\mathbf{s'}_2 = \{t_4, t_5, t_6\}$.

Figure D.10: Structural analysis of R'.

The overall net preserves the internal conservative component $\mathbf{f}_{c1}$ (Example D.3) through $\mathbf{f'}_1$ and $\mathbf{f'}_2$ and it has a new conservative component $\mathbf{f'}_3$. All places belong to at least one conservative component, so the net is bounded.

The Petri net R' preserves the internal repetitive component $\mathbf{s}_c$ through $\mathbf{s'}_1$ and $\mathbf{s'}_2$ in FIG. D.10. No new repetitive component emerges in the overall net. All transitions belong to at least one repetitive component, meaning no conclusions can yet be drawn about liveness. The construction of the marking graph shows that the net is live and reversible.

For the marking $M' = p_3 p_4 p_7$ the (ordered) sequences $s'_1 = t_1 t_2 t_3$ and $s'_2 = t_4 t_5 t_6$ can be fired, making them realizable t-invariants. The place invariants for this marked Petri net are as follows:

$I'_{c_1} : M(M3) + M(C3) + M(S3free) = 1,$
$I'_{c_2} : M(M5) + M(C5) + M(S5free) = 1,$
$I'_{c_3} : M(M3) + M(M5) + M(CellFree) = 1.$

The preserved place invariants and realizable t-invariants guarantee that the behavior of the vehicles in each section is correct. The place invariant I'_{c_3} guarantees that there is at most one vehicle in the two crossing sections.

This illustrates a bottom up approach consisting in introducing behavioral constraints step by step in order to build a more complex model.

◇

Appendix E

TINA TIME PETRI NET ANALYZER

TINA (TIme petri Net Analyzer) is a toolbox for the editing and analysis of ordinary Petri Nets and Time Petri Nets, and has been developed at LAAS/CNRS. More information and download can be found at https://projects.laas.fr/tina//home.php.

E.1 State Space Analysis

The graph of reachable markings, defined in Section 2.1.9, can be obtained by TINA in two versions, textual and graphical. In both cases, choose Menu Tools/State Space Analysis. A window pops out, as shown in FIG. E.1. In the field "building" (left upper corner) choose "marking graph (-R)".

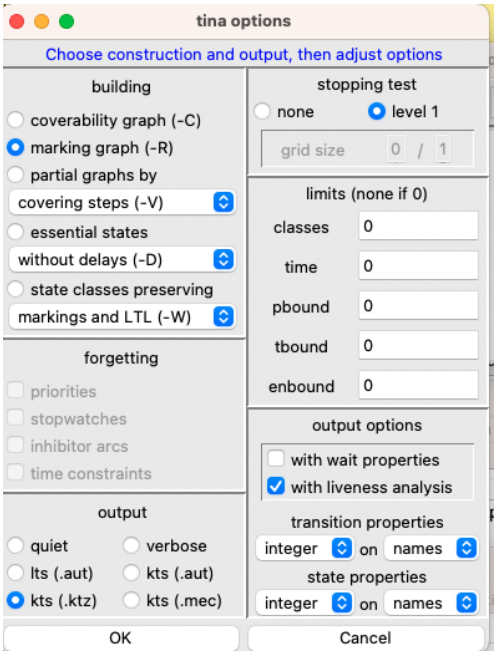


Figure E.1: Graphical version.

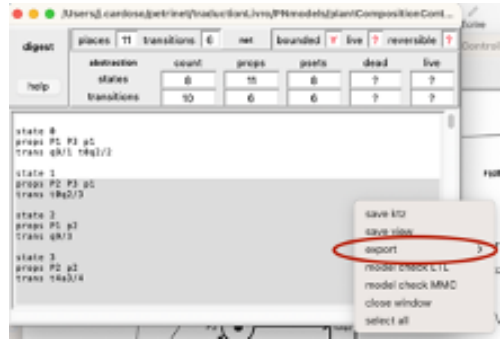


Figure E.2: Choose Export/ aut(lts).

For a textual version of the marking graph, with markings contents, choose "verbose" in the field "output" (left bottom corner) choose and click OK.

For a graphical version of the graph, choose "kts (.ktz)" in the field "output" (left

bottom corner). In the field "output options" choose "with liveness analysis" and click OK, as shown in FIG. E.1.

A new window pops out, as the one in FIG. E.2. Inside the frame with the state description, right-click and choose "export", then "aut (lts)". Yet a new window pops out, as the one in FIG. E.3. Inside the frame, right-click and choose "open file in nd". Always inside this window, go to Tina menu and choose Edit/Draw. You don't need to save file .aut. A small window "draw options" pops, choose "neato", click OK, as represented in FIG. E.4. A last window pops out, with the file .adr. Save this file. You can change the place of the states (markings). For knowing the marking contents you need to generate a textual version of the marking graph ("verbose" option in the field "output" in FIG. E.1).

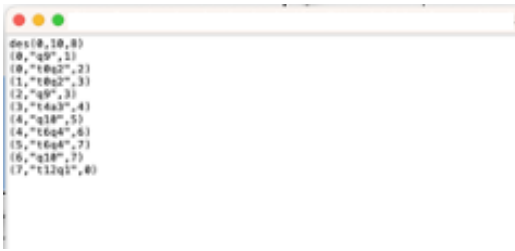


Figure E.3: File .aut.

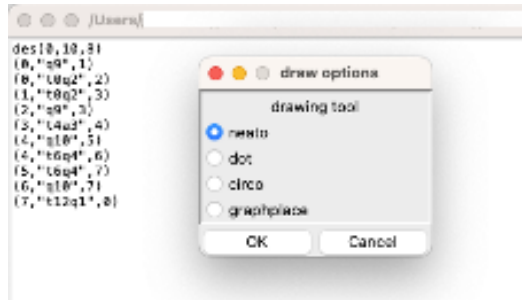


Figure E.4: Open an editor.

In general, nd put the state '0' on the bottom. You can go to menu Edit/flip vert, and the initial state will be on the top. You can reorganise the marking nodes. No need to save .ktz neither.

E.2 Structural analysis

Tina presents the place and transition linear invariants under in the form of p-semi-flow and t-semi-flow generating sets, respectively.

For the example 2 solved in Section 2.5.2 Tinas provides these values:

```
p-semi-flow generating set
p3 p8 (1)
p1 p2 p3 p4 p5 (1)
p5 p7 p9 (1)
p6 p7 (1)
```

```
t-semi-flow generating set
ta tb tc (3)
td te tf (3)
tg th (2)
```

The p-semi-flow `p3 p8` corresponds to the conservative component $\mathbf{f}^3 = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0]^T$. And `p3 p8 (1)` corresponds to the place invariant I_3: $M(p_3) + M(p_8) = 1$. The number in parenthesis corresponds to $\mathbf{f}^{3T} M_0$.

The t-semi-flow `ta tb tc` corresponds to the repetitive component $\mathbf{s}^1 = [1\ 1\ 1\ 0\ 0\ 0\ 0\ 0]^T$. The number in parenthesis indicates the number of transitions in the t-semi-flow.

E.3 Synchronous and Asynchronous Composition

Tina allows both synchronous and asynchronous composition, as seen in Section 3.7.2, using two options:

- **File/Include/as is**: Elements (transitions or places) with the same name are merged automatically. ;
- **File/Include/renamed**: The user manually merges the elements (transitions or places). To merge, move an element e_1 exactly over another element e_2 to be merged. Then right-click on e_1 and check the field "fuse e_1 e_2 as". The name and the label of the merged transition can be changed if needed.

Pay attention when choosing names for the places and transitions in each Petri net model when using the option *File/Include/as is*, as elements with the same name will be merged.

Bibliography

AKERKAR, Rajendra, SAJJA, Priti. Knowledge-based systems. Jones & Bartlett Publishers, 2009.

ANDREU, D., PASCAL, J. C., VALETTE, R. Interaction of discrete and continuous parts of a batch process control system. In: WORKSHOP ON ANALYSIS AND DESIGN OF EVENT-DRIVEN OPERATIONS IN PROCESS SYSTEMS, 1995, London.

ANDREU, D., PASCAL, J. C., PINGAUD, H., VALETTE, R. Batch process modelling using Petri nets. In: IEEE INTERNATIONAL CONFERENCE ON SMC, 1994, San Antonio, USA. 314-319p.

ANDREWS, G. R. *Concurrent programming: principles and practice*. California: The Benjamin/Cummings Publishing Company, Inc., 1987. 637p. ISBN 0-8053-0086-4.

BERTHOMIEU, B., MENASCHE, M. An enumerative approach for analysing time Petri nets. *Information processing*, p. 41-46, 1983.

BERTHELOT, G. *Transformations and decompositions of Nets*. In W. Brauer et al. (eds.), Petri Nets: Central Models and Their Properties Springer-Verlag Berlin Heidelberg 1987.

BOUCHON-MEUNIER, B. *La logique floue*. 2.ed. Paris: Presses Universitaires de France, 1994. 128p.ISBN 2 13 045007-5.

BLANCHARD, M. *Comprendre maîtriser et appliquer le Grafcet*, France: Cepadues Éditions, 1979. 174p.

BRAMS, G.W. *Réseaux de Petri: théorie et pratique*; Tome 1 théorie et analyse. France: Éditions Masson, 1982. 184p. ISBN 2-903-60712-5.

BRAMS, G.W. *Réseaux de Petri: théorie et pratique*; Tome 2 Modélisation et applications. France: Éditions Masson, 1983. 160p. ISBN 2-903-60713-3.

BRAUER W. (Ed.) *Net theory and applications*; Proceedings of the advanced course on general net theory of processes and systems. Hamburg: Springer-Verlag, 1980. (Lecture Notes in Computer Science 84).

BRAUER W. (Ed.): *Petri Nets: central models and their properties*. Hamburg: Springer-Verlag, 1987. (Lecture Notes in Computer Science 254).

BRAUER W. (Ed.): *Petri Nets: applications and relationships to other models of concurrency.* Hamburg: Springer-Verlag, 1987. (Lecture Notes in Computer Science 255).

CAO, T., SANDERSON, A. C. A Fuzzy Petri net approach to reasoning about uncertainty in robotic systems. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1993. *Proceedings of* ... vol. 1, p. 317-322.

CARDOSO, J., VALETTE, R., DUBOIS, D. Petri nets with uncertain markings. In: ROZENBERG, G. (Ed.). *Lecture Notes in Computer Science*; Advances in Petri nets 1990. Berlin: Springer Verlag, 1991. v. 483, p. 64-78.

CARDOSO, J., VALETTE, R. and PRADIN-CHEZALVIEL., B. Linear logic for imprecise firings in fuzzy Petri nets, In B. BOUCHON-MEUNIER, L. ZADEH and R. YAGER (Eds). *Fuzzy logic and soft computing.* World Scientific, 1995. p. 119-128.

CARROL, J., LONG, D. *Theory of finite automata.* Englewood Cliffs, NJ: Prentice-Hall Int. Ed., 1989. 438p. ISBN 0-13-913815-3.

CASSANDRAS, C. G. *Discrete event systems: modeling and performance analysis.* Boston: Aksen Assoc. Inc. Publishers, 1993. 790p. ISBN 0-256-11212-6.

CHEN, Shyi-Ming, KE, Jyh-Sheng, CHANG, Jin-Fu. Knowledge representation using fuzzy Petri nets. *IEEE Trans. on Knowledge and Data Engineering*, local, v. 2, n. 3, p. 311-319, Sept. 1990.

CLARKE JR., Edmund M., GRUMBERG, Orna and PELEG, Doron. MODEL CHECKING. The MIT Press, 330 pp. December 20, 1999. ISBN: 9780262032704.

COURVOISIER, M., VALETTE, R. *Commande des Procédés Discontinus - Logique Séquentielle.* Bordas: Dunod Université, 1986. 181p. ISBN 2-04-016423-5.

DAVID, R., ALLA, H.: *Du Grafcet aux Réseaux de Petri.* Paris: Ãditions Hermes, 1989. 424p.

DAUBAS, B., PAGES, A., PINGAUD, H.: *Combined simulation of hybrid processes.* IEEE INTERNATIONAL CONFERENCE ON SMC, 1994, San Antonio, USA. p. 314-319.

DUBOIS, D., PRADE, H. *Possibility theory: an approach to computerized processing of uncertainty.* New York: Plenum Press, 1988. 263p. ISBN 0-306-42520-3.

FITTING, M. *First-order logic and automated theorem proving.* New York: Springer-Verlag, 1990. 241p.

FREEDMAN, P., MALOWANY A. *Sage: a decision support system for the optimization of repetitive workcell sequencing problems.* Report TR-CIM-88-16, McGill University, 1988.

GENRICH, H. J. predicate/transition nets. In: JENSEN, J. and ROZENBERG, G. (Eds). *High-level Petri nets: theory and application*. Berlin: Springer-Verlag, 1991. p. 3-43. ISBN 3-540-54125-X, ISBN 0-387-54125-X.

GHALLAB, M., DANA N., TRAVERSO, P. Automated planning: theory and practice. *The Morgan Kaufmann Series in Artificial Intelligence*, Morgan Kaufmann, Amsterdam, 2004. ISBN 978-1-55860-856-6.

GIRARD, J.Y. Linear logic. *Theoretical Computer Science*, n. 50, p. 1-102, 1987.

GIRARD, J.Y. La logique linéaire. *Pour la Science*, n. 150, p. 74-85, avril 1990.

KARP, R.M., MILLER, R.E. : *Parallel program schemata*. Journal of Computer and System Sciences, Vol.3, 1969, pp.147-195.

KLIR, G. J., FOLGER, T. A. *Fuzzy sets, uncertainty, and information*. New Jersey: Prentice Hall, 1988. 355p. ISBN 0-13-345984-5.

GONDRAM, M., MINOUX, M. *Graphes et algorithmes*. Paris: Editions Eyrolles, 1985. 545p.

HACK, M. H.-T. *Analysis of production schemata by Petri nets*. Massachussets: MIT, 1972. (Thesis, Master of Science in Computer Science).

HEUSER, C. A. Modelagem conceitual de sistemas: redes de Petri. In: ESCOLA BRASILEIRO-ARGENTINA DE INFORMÁTICA, 5º, 1991, Nova Friburgo. Campinas: R. Vieira Gráfica e Editora Ltda, 1991. 150p.

HOLT, A., COMMONER, F. *Events and conditions*. Massachussets: Record of the Project MAC Conference on Concurrent Systems and Parallel Computation (MIT), June 1970. p. 3-33.

INTERNATIONAL CONFERENCE ON APPLICATION AND THEORY OF PETRI NETS, 1980–. *Proceedings ....* (The best papers are submitted and published by Springer-Verlag in the series *Lecture Notes in Computer Science: Application and Theory of Petri nets*).

INTERNATIONAL WORKSHOP ON PETRI NETS AND PERFORMANCE MODELS, 1985–. *Proceedings ....* Whashington: IEEE Computer Society Press.

JENSEN, K. A high-level language for system design and analysis. In: JENSEN, J. and ROZENBERG, G. (Eds). *High-level Petri nets: theory and application*. Berlin: Springer-Verlag, 1991. p. 44-119. ISBN 3-540-54125-X, ISBN 0-387-54125-X.

JOHNSONBAUGH, R. *Discrete mathematics*. 3.ed. New York: Macmillan Publishing Company, 1993. 800p.

LOONEY, C. G. Fuzzy Petri nets for rule-based decision making. *IEEE Trans. on Systems Man and Cybernetics*. v. 18, n. 1, p. 178-183, jan./feb. 1988.

MAZIERO, C. A. *Um ambiente para a análise e simulação de sistemas modelados por redes de Petri*. Florianópolis: Universidade Federal de Santa Catarina, 1990. 96p. (Dissertação Mestrado em Sist. Cont. e Automação Industrial).

MERLIN, P. *A study of the recoverability of computer systems*. Irvine: University of California, 1974. (Thesis, Phd in Computer Science).

MIYAGI, P. E. *Controle programável: fundamentos do controle de sistemas a eventos discretos*. São Paulo: Editora Edgard Blücher Ltda., 1995. 204p.

MOLLOY, M.: *On the integration of delay and throughput measures in distributed processing models*. Report csd-810021, University of California, 1981.

MURATA, T., ZHANG, D. A predicate-transition net model for parallel interpretation of logic programs. *IEEE Transaction on Software Engineering*, v.14, n.4, p. 481-497, april 1988.

MURATA, T. Petri nets: properties, analysis and applications, *Proceedings of the IEEE*, local, v.77, n.4, p. 541-580, april 1989.

OLIVEIRA, F. A. S. *Um ambiente integrado de ferramentas para o projeto de sistemas a eventos discretos*. Florianópolis: Universidade Federal de Santa Catarina, 1994. 84p. (Dissertação, Mestrado em Sist. Cont. e Automação Industrial).

PRADIN CHÉZALVIEL, B., VALETTE, R. Petri nets and Linear logic for process oriented diagnosis, In: *IEEE/SMC INT. CONF. ON SYSTEMS, MAN AND CYBERNETICS: SYSTEMS ENGINEERING IN THE SERVICE OF HUMANS*, 1993, Le Touquet, France. *Proceedings of ....* v.2, p.264-269.

PETERSON, J.L. *Petri net theory and the modelling of systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981. 286p.

PETRI, C. A. Communication with automata. Supplement 1 to technical report RADC- TR-65-377, Vol 1, New-York 1966, Translated from Kommunikation mit Automaten PhD Bonn 1962.

RAMCHANDANI, C.: *Analysis of asynchronours concurrent systems by timed Petri nets*. Project mac tr 120, Massachussets Institute of Technology, 1974.

REISIG, W. *Petri nets: an introduction*, Berlin: Springer Verlag, 1985. 161p. (EATCS Monographs on Theoretical Computer Science). ISBN 0-387-13723-8.

REUTENAUER, C. *The mathematics of Petri nets*, Hertfordshire: Prentice Hall Int. Ltda, 1990. 117p.

ROSEN, K. H. *Discrete mathematics and its applications*. 2.ed. local: McGraw-Hill Int. Editions, 1991. 793p. ISBN 0-07-112788-7.

SCARPELLI, H., GOMIDE, F., YAGER, R. A Reasoning algorithm for high level fuzzy Petri nets, *IEEE Trans. on Fuzzy Systems*, v.4, n.3, p.282-294, August 1996.

SIBERTIN-BLANC, C. High-level Petri nets with data structures. In: EUROPEAN WORKSHOP ON APPLICATION AND THEORY OF PETRI NETS, 6º, 1985, Helsinki, Finland. Helsinki: Ed. Digital Systems Laboratory, p.141-170.

SILVA, M.: *Las redes de Petri en la automática y la informática*. Madri: Ed. AC, 1985. 190p.

SILVA, M., VALETTE, R. Petri nets and flexible manufacturing. In: GOOS, G., HARTMARIS, J.: *Lecture Notes in Computer Science*; Advances in Petri nets. Berlin: Springer Verlag, 1990. v. 424, p. 374-417.

SOARES, F. A., CARDOSO, J., CURY, J. E. An integrated environment of tools for the design of manufacturing systems, In: WORKSHOP INTELLIGENT MANUFACTURING SYSTEMS, 1994, Viena, Austria.

TINA. https://projects.laas.fr/tina/index.php

TSAI, Wei-Tek , VISHNUVAJJALA, Rama, ZHANG, Du. Verification and validation of knowledge-based systems. In. IEEE trans. on knowledge and data engineering, V 11, N 1, pg 202-212, 1999.

VALETTE, R., ATABAKHCHE. H. Petri nets for sequence constraint propagation in knowledge based approaches. In: VOSS, K., GENRICH, H. J., ROZENBERG, G.: *Concurrency and Petri Nets*, Berlin: Springer Verlag, 1987. p. 555-569. ISBN 3-540-18057-5.

VALETTE, R., ANDREU, D., CARDOSO, J., PASCAL, J.C. Fuzzy Petri nets and their application in CIME, Special Issue on Recent Applications of Petri Nets of the *Transactions of the Institute of Electrical Engineers of Japan*, v. 114-C, n. 9, p. 876-880. 1994.

ZISMAN, M.D. Use of production systems for modeling asynchronous concurrent processes. In: WATTERMAN, D.A., HAYES-ROTH, F. (Ed.). *Pattern Directed Inference systems*. London: Academic Press, 1978, p. 53-68.

ZURAWSKI, R., ZHOU, M. Petri nets and industrial applications: a tutorial, *IEEE Trans. on Industrial Electronics*, v.41, n.6, p. 567-583, 1994.

ZADEH, L.A. Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and Systems*, n. 1, p. 3-28, 1978.

ZADEH, L.A. Theory of fuzzy sets. In BELZER, J., HOLZMAN, A., KENT A. (Ed.), *Encyclopedia of Computer Science and Technology*, New York, 1977.